COMBATTERE LO SPAM
LE TECNICHE ED IL CODICE PER RIPULIRE LE MAIL

VERSIONE PLUS RIVISTA+LIBRO+2CD €14,90
VERSIONE STANDARD RIVISTA+2CD €6,90
Periodicità mensile • MAGGIO 2004 • ANNO VIII, N.5 (80)
Poste Italiane • Spedizione in a.p. - 45% • art. 2 comma 20/b legge 662/96 - AUT. N. DCDC/033/01/CS/CAL

ioPROGRAMMO

IN ALLEGATO

CODICE SOTTO CHIAVE!

Come impedire l'utilizzo illecito del software che sviluppiamo

- **Hardware** Guida super-pratica alla costruzione di una chiave di protezione
- **Software** Come realizzare applicazioni che si attivano via Internet

CONTROLLO REMOTO IN VISUAL BASIC 6.0

Creiamo un'applicazione per gestire a distanza tutte le funzioni del PC

GPS E TELEFONINI
Il codice per far dialogare un GPS con un server, tramite un cellulare GPRS

OFFICE 2003 E XML
Automatizzare la gestione e la presentazione dei contenuti con XSL

EXPLOIT
Come fanno gli hacker a mandare in crash Windows 2000 e XP da remoto

SISTEMA
- Sviluppiamo un tool per realizzare fotomosaici
- La gestione completa dei Fax

INTERNET
- Java: la mail è servita
- JSP: i custom tag
- Web Services: invio e ricezione di file binari

CORSI
- C++: ottimizzazione del codice
- Delphi: comprendere la programmazione a oggetti, disegnando
- Java: la gestione della memoria

ADVANCED
- String matching con gli automi a stati finiti
- Da UML a Java: generare il codice automaticamente

ISSN 1128-594X
ioProgrammo (Plus) Anno VIII - N° 5 (80) • € 14,90
EDIZIONI MASTER www.edmaster.it

SKIN! I SOFTWARE CAMBIANO PELLE

Anno VIII - n. 5 (80) Maggio 2004

▼ Facciamoci pagare

Una buona parte di voi lettori ha fatto della programmazione il suo mestiere. Questo mese vi diamo una mano a guadagnare con il prodotto del vostro ingegno. Vi proponiamo due soluzioni per la protezione delle vostre applicazioni dalla pirateria: una meramente software, l'altra che prevede l'impiego di una chiave hardware, ben più difficile da crackare.
Ovviamente, questa operazione non è indirizzata alle multinazionali del software né vuole inserirsi in quella forma di paranoia che vede gli hacker come i nuovi untori. La redazione di ioProgrammo è impegnata da anni nella diffusione della cultura informatica in Italia, attraverso esempi e guide che coprono tutto il ciclo di vita di un'applicazione. È dunque arrivato il momento di occuparci anche di una delle fasi più delicate di questo ciclo: quella che tocca i nostri portafogli!
A questo proposito, vorrei anche preannunciarvi che a breve lanceremo una nuova rubrica di suggerimenti per tutti gli aspetti legali ed economici legati alla attività di sviluppo e distribuzione del software.
Fateci sapere cosa ne pensate!

raffaele@edmaster.it

All'inizio di ogni articolo, troverete un nuovo simbolo che indicherà la presenza di codice e/o software allegato, che saranno presenti sia sul CD (nella posizione di sempre *\soft\codice* e *\soft\tools*) sia sul Web, all'indirizzo http://cdrom.ioprogrammo.it.
Per scaricare software e codice da Internet, ogni mese indicheremo una password differente. Per il numero che avete fra le mani la combinazione è:

Username: **not** *Password:* **alone**

ioProgrammo

Anno VIII - N.ro 5 (80) - Maggio 2004 - Periodicità Mensile
Reg. Trib. di CS al n.ro 593 del 11 Febbraio 1997
Cod. ISSN 1128-594X
E-mail: ioprogrammo@edmaster.it
http://www.edmaster.it/ioprogrammo
http://www.ioprogrammo.it

Direttore Editoriale Massimo Sesti
Direttore Responsabile Romina Sesti
Responsabile Editoriale Gianmarco Bruni
Responsabile Marketing Antonio Meduri
Editor Gianfranco Forlino
Coordinamento redazionale Raffaele del Monaco
Redazione Antonio Pasqua, Thomas Zaffino
Collaboratori M. Autiero, L. Barbieri, M. Bigatti, L. Buono, M. Canducci, M. Del Gobbo, F. Grimaldi, E. Florio, F. Lippo, M. Locuratolo, A. Margarese, A. Marroccelli, F. Mestrone, G. Naccarato, C. Pelliccia, P. Perrotta, D. Pingitore, L. Spuntoni, F. Vaccaro.
Segreteria di Redazione Veronica Longo

Realizzazione grafica Cromatika S.r.l.
Responsabile grafico: Paolo Cristiano
Coordinamento tecnico: Giancarlo Sicilia
Impaginazione elettronica: Aurelio Monaco

"Rispettare l'uomo e l'ambiente in cui esso vive e lavora è una parte di tutto ciò che facciamo e di ogni decisione che prendiamo per assicurare che le nostre operazioni siano basate sul continuo miglioramento delle performance ambientali e sulla prevenzione dell'inquinamento"

Certificato UNI EN ISO 14001
N. 9191 CRMT

Realizzazione Multimediale SET S.r.l.
Coordinamento Tecnico Piero Mannelli
Realizzazione CD-Rom Paolo Iacona
Pubblicità Master Advertising s.r.l.
Via Cesare Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121207
e-mail advertising@edmaster.it
Sales Director: Max Scortegagna
Segreteria Ufficio Vendite Daisy Zonato

Editore Edizioni Master S.r.l.
Sede di Milano: Via Cesare Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121206
Sede di Rende: C.da Lecco, zona industriale - 87036 Rende (CS)
Amministratore Unico: Massimo Sesti

Abbonamento e arretrati
ITALIA: Abbonamento Annuale: ioProgrammo Basic (11 numeri) + mini hub USB: €55,50 sconto 27% sul prezzo di copertina di €75.90
ioProgrammo Libro (11 numeri + libro) + mini hub USB: €90,50 sconto 27% sul prezzo di copertina di €123.90
Offerte valide fino al 30/06/04 salvo esaurimento scorte
ESTERO: Abbonamento Annuale: ioProgrammo Basic (11 numeri): € 151,80. ioProgrammo Plus (11 numeri + 6 libri): € 257,00
Costo arretrati (a copia): il doppio del prezzo di copertina + € 5.32 spese (spedizione con corriere). Prima di inviare i pagamenti, verificare la disponibilità delle copie arretrate allo 02 831212.
La richiesta contenente i Vs. dati anagrafici e il nome della rivista, dovrà essere inviata via fax allo 02 83121206, oppure via posta a EDIZIONI MASTER via Cesare Correnti, 1 - 20123 Milano, dopo avere effettuato il pagamento, secondo le modalità di seguito elencate:

- **cc/p n.16821878** o **vaglia postale** (inviando copia della ricevuta del versamento insieme alla richiesta);
- **assegno bancario non trasferibile** (da inviarsi in busta chiusa insieme alla richiesta);
- **carta di credito**, circuito VISA, CARTASI', MASTERCARD/EUROCARD, (inviando la Vs. autorizzazione, il numero della carta, la data di scadenza e la Vs. sottoscrizione insieme alla richiesta).
- **bonifico bancario** intestato a Edizioni Master S.r.l. c/o Banca Credem S.p.a. c/c 01 000 000 5000 ABI 03032 CAB 80880 CIN Q (inviando copia della distinta insieme alla richiesta).

SI PREGA DI UTILIZZARE IL MODULO RICHIESTA ABBONAMENTO POSTO NELLE PAGINE INTERNE DELLA RIVISTA. L'abbonamento verrà attivato sul primo numero utile, successivo alla data della richiesta.
Sostituzioni: Inviare il CD-Rom difettoso in busta chiusa a: Edizioni Master Servizio Clienti - Via Cesari Correnti, 1 - 20123 Milano

Assistenza tecnica: ioprogrammo@edmaster.it

Servizio Abbonati:
☎ tel.02 831212
@ e-mail: servizioabbonati@edmaster.it

Stampa: Rotoeffe Via Variante di Cancelliera, 2/6 - Ariccia (Roma)
Stampa CD-Rom: Deluxe Italy S.r.l. - via Rossini, 4 - Tribiano (MI)
Distributore esclusivo per l'Italia: Parrini & C S.p.A.
Via Vitorchiano, 81 - Roma

Finito di stampare nel mese di Aprile 2004

Edizioni Master edita:
Idea Web, Go!OnLine Internet Magazine, Win Magazine, PC Fun extreme, Quale Computer, DVD Magazine, Office Magazine, La mia Barca, ioProgrammo, Linux Magazine, Softline Software World, HC Guida all'Home Cinema, MPC, Discovery DVD, Computer Games Gold, inDVD, I Fantastici CD-Rom, PC VideoGuide, I Corsi di Win Magazine, I Filmissimi in DVD, La mia videoteca, Le Collection, Tv & Satellite.

News

FILEMAKER PRO 7 DISPONIBILE IN ITALIANO

FileMaker Pro 7 e FileMaker Developer 7 sono finalmente disponibili sul mercato in versione italiana. FileMaker 7 offre oltre cento nuove funzioni, una maggiore potenza ed una interfaccia più semplice: la gamma FileMaker Pro 7 si può ritenere la più intuitiva e produttiva versione mai realizzata.
Da rimarcare i nuovi campi Container, attraverso cui è possibile importare, memorizzare ed esportare qualsiasi file o documento: dai file di Word alle presentazioni in Power Point sino alle immagini, ai video, alla musica digitale. Con FileMaker Pro 7 è ora possibile aprire più finestre all'interno di uno stesso database per offrire agli utenti un accesso simultaneo più rapido a diverse viste delle informazioni richieste.

www.filemaker.com

JAVA OPEN SOURCE? DIFFICILE!

Durante una conferenza tenuta il 16 marzo scorso a Washington, McNealy ha definito alquanto inverosimile l'ipotesi che Sun possa rilasciare Java alla comunità Open Source, una ipotesi spinta con forza da IBM. La risposta di McNealy alle pressioni di IBM pare sia stata alquanto seccata: "Quando IBM rilascera DB2 come Open Source, solo allora potrà dirmi cosa fare delle mie attività". McNealy ha comunque confermato che SUN sta studiando l'ipotesi, cercando in particolare di comprendere quali problemi potrebbero essere risolti, che non siano stati già risolti senza il ricorso all'Open Source.

www.java.sun.com

WORLD WIDE DEVELOPER CONFERENCE DI APPLE

Dal 28 Giugno al 2 Luglio si terrà la conferenza mondiale di Apple per gli sviluppatori: organizzata dalla casa di Cupertino, si svolgerà a San Francisco, California.

Si tratta di un evento dedicato a tutti gli sviluppatori Mac OS X, che avranno la possibilità di confrontarsi con ingegneri Apple a altri programmatori provenienti da ogni angolo del pianeta. Di seguito sono elencate alcune delle sessioni dell'evento:

- Application Technologies
- Development Tools
- Enterprise IT
- Graphics and Media
- Hardware Technologies
- OS Foundations
- QuickTime and Digital Media

Inoltre saranno affrontati tempi che spaziano dalla sicurezza, crittografia, autenticazione, file system, I/O, all'implementazione Apple del server X11.
Tutte le informazioni al momento disponibili sono sul sito al seguente url:
http://developer.apple.com/wwdc/

per procedere alla registrazione (entro il 30-04-2004):
http://developer.apple.com/wwdc/aregistration.html

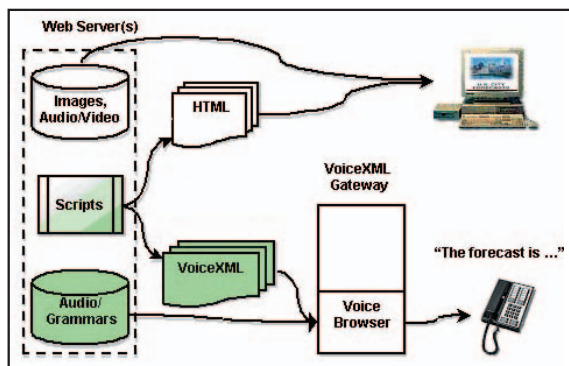
VOICEXML 2.0 È STANDARD

Il World Wide Web Consortium ha rilasciato due nuove raccomandazioni per il suo Speech Interface Framework, con lo scopo di rendere le informazioni presenti su Internet ai miliardi di persone connesse a telefoni fissi e cellulari. La versione 2.0 di VoiceXML è stata progettata per permettere un utilizzo totale dell'interfaccia telefonica: riconoscimento e sintesi vocale, supporto dei tasti a toni (DTMF), registrazione del parlato e telefonia generica.
L'obiettivo è consentire

VIDEOGIOCHI SENZA FRONTIERE

Microsoft ha annunciato XNA, una nuova piattaforma di sviluppo orientata ai videogiochi. Nata allo scopo di permettere il controllo degli ambienti più complessi, XNA permette di sviluppare giochi e applicazioni multipiattaforma, supportando contemporaneamente Windows, Xbox e tutti i dispositivi basati su Windows Mobile. In XNA confluiranno numerosi tool, anche di terze parti: pur salvaguardando una grande varietà di scelta per gli strumenti, sarà garantita la coerenza necessaria a semplificare il ciclo di sviluppo.
Del pacchetto farà parte Xbox Live, che diventerà dunque disponibile anche per gli sviluppatori di applicazioni per Windows, semplificando così la realizzazione e la distribuzione di applicazioni on-line.

Comuni a tutte le piattaforme saranno i controlli che intercettano l'input degli utenti: le stessi API e gli stessi pulsanti saranno disponibili sia per Xbox, sia per Windows.
Più numerose di tutte sono le novità nel campo della grafica e dell'audio: nuovi tool e nuovi linguaggi renderanno più rapido il passaggio dalla fase creativa alla realizzazione del codice.

www.microsoft.com/xna/

Un possibile scenario per VoiceXML.

lo sviluppo di applicazioni Web-Based che utilizzino il telefono per interagire e fornire informazioni agli utenti.
La seconda raccomandazione, Speech Recognition Grammar Specification Version 1.0, svolge una funzione chiave all'interno di VoiceXML, fornendo agli sviluppatori la possibilità di descrivere le possibili risposte degli utenti.
Questa seconda raccomandazione definisce una sintassi per rappresentare le grammatiche da utilizzare nel riconoscimento del parlato: gli sviluppatori potranno specificare le parole e le strutture lessicali necessarie ad un corretto funzionamento del riconoscitore vocale.
La sintassi con cui è possibile specificare le grammatiche sono presentate in due forme: *Augmented BNF Form* e *XML:* si avvicina l'era di un web vocale?

www.w3.org

HACKING IN AULA

Spoofing, Man in the Middle, fingerprinting, social engineering sono tutti termini ancora oscuri per molti, anche se la loro conoscenza è un obbligo per chiunque oggi si interessi di networking e in particolare di sicurezza. Per affrontare questi temi e per entrare in profondità nel mondo dell'hacking dei network, gli autori dell'opera "*Hacker@tack*" e la società di formazione SPC Italia hanno appena lanciato un nuovo programma di corsi di formazione dedicati all'hacking. Un modo diverso di affrontare il tema della sicurezza informatica che parte da quello che molti considerano il suo "lato oscuro". Oltre a un corso introduttivo di quattro giornate che affronterà il tema dell'hacking nei suoi aspetti complessivi (dalle motivazioni che portano a un attacco, fino alle tecniche più diffuse e alle azioni preventive) ne sono stati previsti altri specifici dedicati ai tool più utilizzati dagli hacker, le problematiche degli attacchi via Web e di quelli a firewall e router. A completamento dei percorsi formativi sono stati previsti anche corsi sull'information security management sia dal punto di vista tecnico sia da quello delle procedure. I corsi si terranno a partire dal mese di maggio a Milano. Il calendario corsi completo è disponibile sul sito. Per ulteriori informazioni telefonare allo 02-66985681 oppure inviare una email a *laura.delgaudio@spcitalia.it*.

www.spcitalia.it.

ECCO I CAMPIONI DELLE OLIMPIADI ITALIANE DELL'INFORMATICA

Premiati a Trento i migliori classificati nella gara che si è tenuta a Marzo. Fra essi i candidati alla squadra italiana per le Olimpiadi Internazionali dell'Informatica che avranno luogo il prossimo settembre in Grecia. Sono stati quaranta i giovani che hanno primeggiato alle selezioni nazionali per le Olimpiadi Internazionali dell'Informatica (IOI) guadagnandosi così il titolo di "Olimpionici Italiani": le selezioni nazionali per le IOI sono infatti denominate Olimpiadi Italiane dell'Informatica (OII). Le OII, patrocinate dal Ministero dell'Istruzione dell'Università e della Ricerca e, quest'anno, dalla Provincia Autonoma di Trento, hanno coinvolto gli 80 studenti provenienti da tutta Italia che hanno primeggiato tra i 10.000 studenti iscritti alle selezioni per l'edizione 2004 dell'evento lo scorso ottobre. La premiazione dei campioni italiani d'informatica 2004 è avvenuta alla presenza del Presidente della Provincia Autonoma di Trento Lorenzo Dellai, dell'assessore alle Politiche giovanili e all'Istruzione per il comune di Trento, del Direttore Generale per l'innovazione tecnologica del MIUR Alessandro Musumeci, del Direttore Generale del Ministero dell'Innovazione Tecnologica Roberto de Marco, del Presidente AICA Ivo De Lotto e del Preside della Facoltà di Scienze dell'Università di Trento. Ai migliori 43 sono state assegnate medaglie (9 oro, 12 argento e 22 bronzo) come avviene nelle competizioni olimpiche tradizionali. I quaranta studenti premiati sono i migliori dei circa 10.000 (di circa 480 Istituti di tutta Italia) che hanno partecipato quest'anno alle precedenti fasi di selezione, a conferma del crescente successo che l'iniziativa sta riscontrando presso la scuola.

www.olimpiadi-informatica.it

Le novità dal mondo Sun

All'ultima conferenza stampa della Sun è stato confermato l'impegno della società americana sul fronte della ricerca e dello sviluppo di sistemi informatici sempre più efficaci e convenienti.

di Federico Mestrone

Il 17 febbraio a Milano, si è tenuto il quinto appuntamento di una serie di scadenze quadrimestrali con cui la Sun intende tenere informato ed aggiornato il pubblico sui propri obiettivi e sui risultati man mano raggiunti. Già in occasione degli incontri precedenti, la compagnia aveva esplicitato la sua intenzione di contare molto sulla ricerca, ed anche in quest'ultima conferenza ha dimostrato la serietà dei suoi intenti: nonostante il periodo che stiamo vivendo sia abbastanza critico per il settore delle tecnologie informatiche, la casa americana crede fermamente che, nel lungo termine, lo studio e la messa a punto di sistemi sempre più sicuri, stabili e semplici da gestire sia la strada più vantaggiosa. La filosofia sottostante a questa strategia è quella di fidelizzare la clientela con soluzioni affidabili, scalabili e sicure che non diano problemi, piuttosto che puntare sulla fornitura di servizi e consulenze per la risoluzione dei problemi.
In questo senso la società propone i suoi prodotti non come singoli componenti da aggregare, ma sempre come parte di un sistema completo in cui le varie parti si integrano ed interoperano ai fini di un deployment il meno problematico possibile: il ruolo della Sun in tutto questo è quello di fornire i vari pezzi del puzzle garantendo la loro funzionalità come unità e come parte di soluzioni più complesse, agendo in complementarità con i propri partner (la iForce Initiative) nello studiare, insieme al cliente, quali siano le alternative di aggregazione più consone alle esigenze da soddisfare.

LE NOVITÀ

In questo quadro, in occasione dell'incontro con la stampa, la Sun ha presentato i più recenti frutti del suo lavoro. È stata annunciata innanzitutto l'uscita sul mercato del nuovo processore *UltraSparc IV*, che permette elevate prestazioni a costi accessibili. La ricerca che ha condotto a questo prodotto ha rivelato che i processori passano molto più tempo ad accedere alla memoria che ad eseguire operazioni: una nuova tecnologia utilizzata dalla Sun per l'UltraSparc IV, detta *Chip MultiThreading* (CMT), sfrutta i tempi di latenza di accesso alla memoria per permettere al processore di portare avanti parallelamente più thread di esecuzione (attualmente due, ma si parla già di quattro thread per il prossimo futuro). Questo ha portato ad un miglioramento delle prestazioni di quasi il 100%, unito ad un costo competitivo ed ad una compatibilità completa con il sistema precedente, al punto che si possono tenere sulla stessa macchina delle board con UltraSparc III insieme ad altre con il nuovo UltraSparc IV.

Fig. 1: La nuova generazione di processori UltraSparc IV.

È stata sottolineata l'importanza che la società dà alla backward compatibility, per garantire quella che è stata definita "protezione degli investimenti", senza costringere i clienti a disfarsi di tutto quanto è vecchio per passare a ciò che è nuovo, ma permettendo loro una passaggio graduale alle nuove soluzioni.
Anche relativamente alla nuova versione del sistema operativo Unix della casa informatica si è parlato di compatibilità: Solaris 10, infatti, offrirà una compatibilità binaria al 100% con Solaris 9. L'uscita di questa nuova versione è prevista nei prossimi mesi, e tra le caratteristiche più evidenti abbiamo la capacità di Self Healing ("auto guarigione" letteralmente), dovuta ad una serie di sonde software che testano e verificano continuamente l'integrità del sistema; una più sofisticata attenzione alla sicurezza e l'integrazione nell'edizione base del sistema operativo delle funzionalità dell'edizione *Trusted* della versione antecedente, il che offrirà per esempio la gestione dei diritti d'accesso a livello di processo; infine si è fatto riferimento agli *N1 Grid Containers*, un approccio innovativo alla virtualizzazione offerto da Solaris 10, che permette di creare partizionamenti software della macchina, condividendo un'unica istanza del sistema operativo, garantendo così da una parte i vantaggi della separazione in comparti stagni del codice eseguito, e al stesso tempo un risparmio delle risorse del computer. Si tratta insomma di un sistema operativo

che promette più up-time, più sicurezza, più affidabilità e una maggiore facilità di gestione, ma sempre ai soliti costi vantaggiosi. Per chi volesse assaporare in anticipo tutti questi vantaggi,
Solaris 10 è disponibile come Early Access per il download dal sito della Sun: ottenere le immagini dei CD di installazione come preview è gratuito per tutti coloro che possiedono una licenza Solaris grazie al programma Solaris Express. Infine, anche se forse un po' scontato, è sempre importante ricordare che oltre alla piattaforma Sparc a 64bit (con compatibilità per l'esecuzione di tutte le applicazioni a 32bit), Solaris 10 è disponibile anche per i PC x86 a 32bit.

	Sun Fire V20z	IBM eServer 325	Dell PE3250	HP rx2600	HP DL360 G3
Processore	AMD Opteron	AMD Opteron	Itanium 2	Itanium 2	Xeon
Dimensione	1U	1U	2U	2U	1U
Memoria (Max)	16GB	12GB	16GB	24GB	8 GB
Distribuzioni Unix/Linux	Solaris, Red Hat EL 3.0 32 and 64 Bit, SuSE ELS 8 AMD, SuSE 9 Pro 64 AMD	SuSE Pro 8.2, SuSE ELS8 AMD	Red Hat AS/WS 2.1 64 bit	HPUX, Red Hat AS2.1, SuSE ELS 8	SuSE LES 8, Red Hat AS/ES 2.1
Supporto nativo App. 32/64 bit	Si	Si	No	No	No
Prezzo* (€)	3,850[1]	4,613[2]	9,179[3]	8,682[3]	5,545[4]

*Prezzi aggiornati al 10 Febbraio 2004

Fig. 3: Una delle tabelle comparative presentate alla conferenza.

PICCOLE RIVOLUZIONI IN FASCIA SERVER

Restando in ambito x86, la Sun aveva annunciato a novembre, in sede del precedente incontro, una partnership con AMD per la creazione di una piattaforma a 64bit con lo slogan "extreme performance at compelling prices".
E dopo pochi mesi segue l'uscita sul mercato di Opteron, il nuovo processore della AMD a 64bit, già integrato nell'ultimissimo server a basso costo della Sun chiamato *Sun Fire V20z*: si tratta di un server ad elevate prestazioni, in grado di offrire non solo totale compatibilità per l'esecuzione di applicazioni a 32bit, ma anche un notevole vantaggio di performance nell'eseguirle. Tale compatibilità e vantaggio si estendono non solo agli applicativi, bensì anche al sistema operativo.

È fornito totale supporto sia per Solaris che per Linux, e RedHat e Suse sono già pronte con versioni a 64bit per Opteron delle loro distribuzioni.
Questa macchina sarà in grado di indirizzare fino a 16GB di RAM, è attualmente disponibile nella versione biprocessore, ma sarà presto in vendita una formula con quattro processori. Supporta la configurazione in daisy chain con altri server, ed offre tre interfacce di rete separate, due per i dati (10/100/ 1000Mbps) e l'altra per la gestione (10/ 100Mbs). Il prezzo di base si aggira poco sotto i € 3.000,00 facendo di questo prodotto una soluzione interessante anche per le piccole imprese o i professionisti che necessitino di macchine performanti ed affidabili.
Per concludere la sintesi dell'incontro, non posso non menzionare il Sun Java Enterprise System, un prodotto basato su Java destinato alle grandi imprese con application server, instant messaging, collaboration, portal server che si propone come stabile ed efficiente alternativa ai più costosi prodotti della competizione, ed il *Java Desktop System*, con cui la Sun vuole offrire un'alternativa a Windows come ambiente grafico di gestione del PC. Il sistema operativo sottostante sarà Linux, con Java come piattaforma esecutiva di un architettura a finestre di grande impatto grafico e dalle promettenti capacità: sarà dotata di tutto il software che ci si aspetta da una GUI complessa per l'utilizzo domestico o di piccola impresa, a partire dall'applicazione di gestione della posta elettronica, dei contatti e degli appuntamenti, ad un browser web firmato Sun, fino alla suite *StarOffice* per la creazione di documenti di testo, spreadsheet, presentazioni, il tutto compatibile con Microsoft Office. Non mancano una serie di tool per la gestione del sistema (console di comandi, gestione dischi, task, e via dicendo).

Fig. 2: Il nuovo server a basso costo di SUN: Sun Fire V20z.

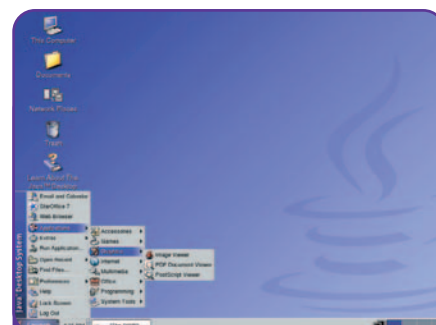



Fig. 4: Una schermata del Java Desktop System.

Ovviamente il costo è di gran lunga inferiore a quello di Windows (si parla di $100, ma in promozione fino a giugno 2004 per soli $50). Ed ora non ci resta che vedere con cosa la Sun sarà in grado di stupirci la prossima volta!

SPECIALE MICROSOFT

ASP.NET Resource Kit

Queste mese, trovate un secondo CD allegato alla rivista: una incredibile quantità di materiale per iniziare a sviluppare le vostre applicazioni ASP.NET!

di Federico Mestrone

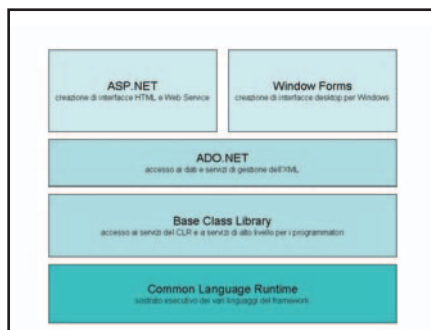
Tutti conoscete sicuramente il framework .NET della Microsoft. Si tratta del nuovo sostrato di sviluppo che la casa di software americana ha architettato per permettere una creazione più rapida di applicazioni sempre più complesse e funzionali. Per chi non lo usa abitualmente, diamo una breve idea della sua struttura (vd anche Fig. 1): alla base il *Common Language Runtime* (CLR) mette a disposizione un ambiente esecutivo per il codice scritto dai programmatori in uno dei linguaggi supportati dal framework (gli ormai famosi C#, JScript, VB.NET e C++ con Managed Extensions, ma la lista non è chiusa…) e funge da collante per l'interoperabilità tra di essi, facendo sì che sia indifferente la scelta di linguaggio che il programmatore fa: garantisce inoltre servizi tipo threading, gestione della memoria, compilazione ed esecuzione del codice, sicurezza delle chiamate. Ad un livello superiore, un set di API orientato agli oggetti (la *Base Class Library*) offre l'accesso a servizi essenziali del sistema operativo, quali l'I/O, gestione di stringhe e collezioni di oggetti, della memoria, delle periferiche principali, della rete, e permette lo sviluppo di vari tipi di progetti applicativi: applicazioni di console, grafiche, servizi web con XML, servizi di Windows, applicazioni per il web.

Fig. 1: La struttura del framework di sviluppo .NET.

UN AMBIENTE GRATUITO

Al di sopra della *Base Class Library*, il framework fornisce anche il supporto per alcune funzionalità di sostegno allo sviluppo software: l'accesso ai dati su database di vario genere viene fornito ad opera di ADO.NET, la nuova generazione di ADO che oltre a mettere a disposizione un'interfaccia comune di recupero e gestione di dati dalle fonti più diverse permette anche di manipolare facilmente l'XML; dal lato invece delle interfacce utente, le applicazioni classiche per windows vengono sviluppate utilizzando Window Forms, una serie di oggetti per la creazione e manipolazione di finestre, mentre le applicazioni web vengono create a partire da ASP.NET, una serie di classi ed interfacce che da un lato danno l'appoggio per la creazione rapida di sofisticate applicazioni basate su browser ed HTML, dall'altro agevolano la creazione di Web Service basati sugli standard universali WSDL, UDDI e SOAP. Nel secondo CD che ioProgrammo mette a disposizione dei suoi lettori questo mese, troverete una quantità enorme di informazioni, tool, controlli aggiuntivi, offerte speciali ed esempi relativi allo sviluppo di siti dinamici e web service con ASP.NET e accesso dati ad opera di ADO.NET.

Il materiale offerto si adatta sia ad utenti già pratici delle due tecnologie che ad utenti che invece vi si avvicinano per la prima volta. Come strumento di sviluppo per la stesura del codice e la creazione di applicazioni è possibile utilizzare il classico Visual Studio .NET (giunto ad oggi alla versione 2003), che come saprete è il prodotto commerciale ufficiale per la produttività sotto il framework .NET. In alternativa, incluso nel CD di ioProgrammo, viene presentato WebMatrix, un software gratuito che consente di creare applicazioni Web in C#, VB.NET o J#, gestire fonti dati su Access o SQL Server (anche nella versione desktop MSDE, fornita nel CD) e soprattutto testare e fare il debug del proprio codice anche senza IIS installato: infatti esso già contiene una versione integrata di Cassini, un web server che permette di eseguire codice ASP.NET, anch'esso completamente gratuito e distribuito con codice sorgente C# all'indirizzo *www.asp.net/Projects/ Cassini/Download/Default.aspx*.

WebMatrix (vd. Fig. 2) offre la possibilità di gestire la pagina come HTML, di vederne solo le porzioni di codice, o vedere tutto insieme, e permette anche di creare

l'HTML in *Design View*, lavorando cioè con un editor *WYSIWYG* che facilita la

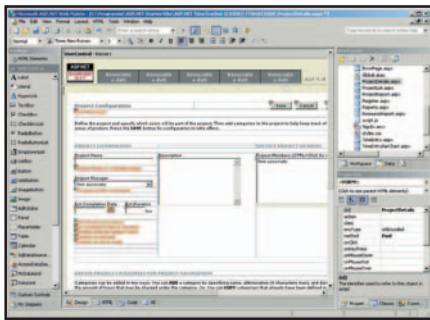



***Fig. 2:* WebMatrix, un tool di sviluppo gratuito per ASP.NET.**

produzione del layout. Potete lavorare con progetti locali, tramite shortcut alle cartelle che li contengono oppure progetti remoti gestibili tramite connessione FTP. Potete inserire sulla pagina controlli web di ASP.NET o semplici controlli HTML tramite un banale drag-and-drop dalla toolbox e creare snippet di codice – anch'essi da trascinare sulla pagina – per aumentare la vostra produttività come programmatori (manca ahimé, qualunque tipo di feature di Statement Completion durante la stesura manuale del codice).

GLI STARTER KIT

Oltre a questo tool di sviluppo, per avere un'idea delle potenzialità delle due tecnologie basate su .NET, sono messe a disposizione dei programmatori una serie di applicazioni già pronte (i cosiddetti *Starter Kit*) complete di codice sorgente dettagliatamente commentato, riutilizzabile e ridistribuibile.
Si tratta di *"Time Tracker"*, per tenere traccia dei tempi di gestione e lavorazione associati ad un progetto; *"Reports"*, una serie di esempi di generazione di grafici di vario tipo; *"Community"*, un sito con forum, che permette lo scambio di idee, informazioni ed opinioni, la condivisione di file, e la gestione di comunità online in generale; *"Commerce"*, un sito di commercio elettronico, dal catalogo, alla registrazione utente, al carrello della spesa; ed infine *"Portal"* (vd Fig. 3), una serie di dieci moduli combinabili per la creazione di portali web con la possibilità di svilupparne di personalizzati, il tutto condito da un tool di amministrazione web che consente all'utente finale con i giusti privilegi di esecuzione di selezionare il formato ed i contenuti preferiti.
Infine, al di là di molto materiale informativo (sotto forma di tutorial da leggere o presentazioni multimediali), trovate anche alcune offerte commerciali di hosting abbastanza interessanti se ancora non avete un provider dove ospitare le

***Fig. 3:* Lo Starter Kit "Portal" in esecuzione.**

vostre applicazioni .NET (si parla di prezzi intorno ai $10 al mese per una media di 30Mb di spazio disco), ed una serie di controlli aggiuntivi di terze parti, quasi tutti però a pagamento o con sola licenza di sviluppo.

UN BREVE PERCORSO GUIDATO

Da dove cominciare, quindi, per utilizzare questa mole di informazioni e prodotti? Per chi già conosce un po' le tecnologie proposte e possiede Visual Studio .NET, direi che la cosa migliore è sfogliare i vari contenuti in cerca di qualche tutorial interessante per approfondire le proprie conoscenze o esplorare possibilità non ancora intraviste del framework (ad esempio, è presente un'intera sessione dedicata allo sviluppo di applicazioni per dispositivi mobili, tipo cellulari, PDA e palmari, raggiungibile dalla home page HTML del CD tramite il link *Mobile Applications*).
Sicuramente poi sarà utile installare e studiare le applicazioni *Starter Kit*, per avere un buon punto di partenza già pronto e funzionante per ognuno dei principali tipi di applicazione web che possiate voler implementare. La sezione sui web service, poi, è ricca di informazioni ed esempi che però sono utilizzabili solo con Visual Studio (WebMatrix non è sufficiente).
Se invece siete del tutto nuovi a questo mondo e non sapete dove andare a mettere le mani, seguite la *Step-by-Step Guide for Developers Not Using Visual Studio .NET*, che trovate nel CD al percorso *<cd-drive>:\Setup\Step-by-Step.htm* e che vi guida nell'installazione di ciò di cui avete bisogno per cominciare, tutto rigorosamente dal CD senza necessità (almeno inizialmente) di una connessione ad internet: comincerete con il framework .NET alla versione 1.1, per poi proseguire con WebMatrix, i componenti per gestire database (MSDE, necessario se volete anche installare gli Starter Kit) ed il ridistribuibile di J# (se volete proprio sviluppare con J#, ma ne vale la pena?), con in coda una serie di link a forum, newsletter e siti con risorse addizionali (qui sì che vi serve una connessione ad Internet!). Dopo l'installazione, invece, potete rifarvi al *Guided Tour* di WebMatrix disponibile sul CD (vd Fig. 4) al percorso *<cd-drive>:\GuidedTour\GetStarted\intro.htm* che vi da una rapida visione del prodotto, e soprattutto vi introduce al mondo ASP.NET con sintassi di base e principi fondamentali (la versione online della guida è raggiungibile tramite il menu di Help di WebMatrix, selezionando *Help*

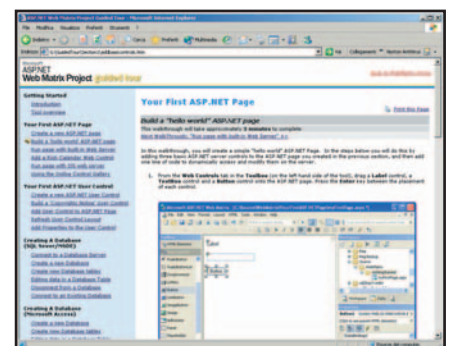



***Fig. 4:* Il Guided Tour all'utilizzo di WebMatrix e ASP.NET - adatto anche a chi comincia da capo!**

Topics e successivamente il link al Guided Tour).
Potete poi installare i vari Starter Kit ed utilizzarli in WebMatrix come base per successivi sviluppi e miglioramenti, potendo così creare applicazioni veramente professionali e complete, il tutto con un framework gratuito ed un tool di sviluppo altrettanto gratuito… ma adesso vi lascio al CD, sperando di avervi dato informazioni a sufficienza per orientarvi un po'!

SOFTWARE SUL CD

Dev-C++ 5.0 beta 8

Un IDE Open Source per lo sviluppo di applicazioni C/C++.

Presentiamo un ambiente integrato per lo sviluppo di applicazioni in linguaggio C/C++ distribuito con licenza GNU GPL. Il programma, di default, utilizza *Mingwn*, port del potente compilatore *GCC* (GNU Compiler Collection), ma può essere utilizzato con *Cygwin* o qualsiasi altro compilatore basato su *GCC*. Dev-C++ consente di creare eseguibili Win32 come applicazioni GUI, console, librerie statiche e DLL.
Dev-C++ è un ambiente di sviluppo completo sotto tutti i punti di vista, tra le caratteristiche principali ricordiamo:

- Supporto per i compilatori basati su GCC (Mingw incluso)
- Debugger GDB (GNU Debugger) integrato
- Supporto multilingue (localizzazione)
- Class browser
- Completamento automatico del codice
- Project Manager
- Evidenziazione della sintassi personalizzabile
- Sistema CVS per il controllo delle versioni
- Wizard per applicazioni a finestre, console, librerie statiche e DLL
- Creazione di *Makefile*
- Compilazione e modifica di file di risorse
- Supporto per la stampa
- Funzioni di ricerca e sostituzione del testo
- Package manager per la gestione di librerie add-on.

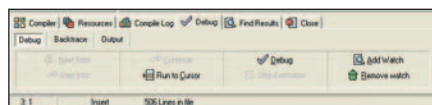
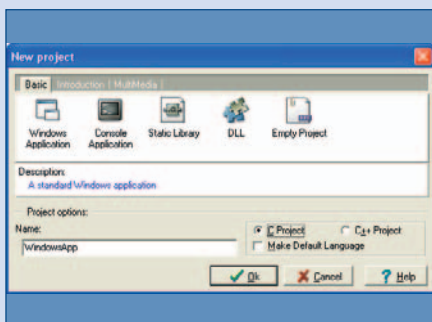
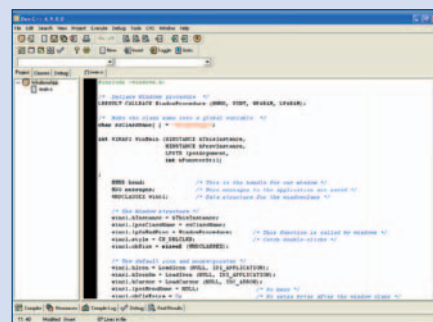
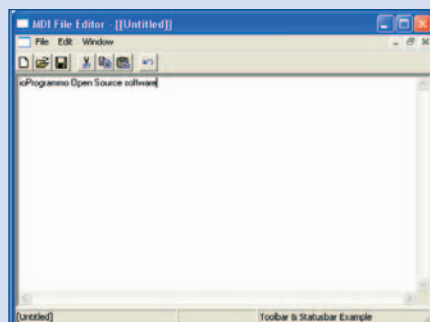



Fig. 1: ***Strumenti per il debugging.***

REQUISITI DI SISTEMA

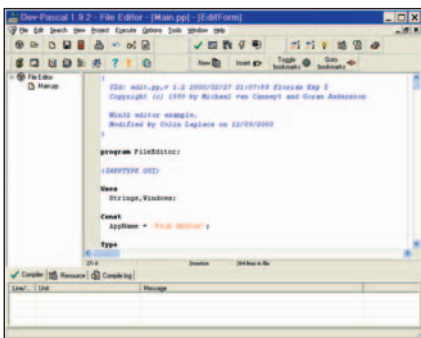
Il programma, veloce e leggero, non necessita di eccessive risorse di sistema (caratteristica non comune a questo tipo di applicazioni), funziona con tutte le versioni di Windows (Win95 e successive), anche se il produttore raccomanda di utilizzare MS Windows 2000 o XP, con almeno 32 MB di memoria RAM, CPU Intel o compatibile con frequenza non inferiore a 400 Mhz e 200 MB di spazio libero su disco.

☑ Dev-C++ 5.0 beta 8
Produttore: Bloodsheed Software
Sul Web: http://www.bloodshed.net
Licenza: GNU GPL
Nel CD: *devcpp4980.exe*

CREARE ED ESEGUIRE UN NUOVO PROGETTO

1 **Selezionare dal menù principale *File/New/Project*, verrà visualizzata la finestra *New project* in cui è possibile selezionare il tipo di applicazione che si vuole creare.**

2 **Premendo il pulsante *OK*, nella finestra *New project*, apparirà la schermata per salvare il file del progetto. Alla fine di questo processo apparirà l'interfaccia principale dell'IDE.**

3 **Per compilare ed eseguire l'applicazione appena creata basta selezionare dal menù principale la voce *Execute/Compile/Run*, oppure, utilizzare i comandi *Compile* e *Run* presenti sulla barra degli strumenti.**

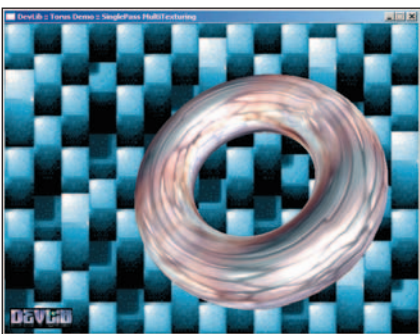
DEV-PAS 1.9.2

Un completo ambiente di sviluppo per Pascal.

Un versatile e potente ambiente di sviluppo integrato per la realizzazione di applicazioni Windows in Pascal.
L'IDE può utilizzare indifferentemente i compilatori Free Pascal e GNU Pascal, entrambi distribuiti con licenza GNU GPL, mentre il codice è disponibile sottoforma di sorgenti Delphi.

Dev-Pas consente di sviluppare qualsiasi tipo di applicazioni, da quelle a console fino alle applicazioni dotate di interfacce grafiche GUI (Graphical User Interface).
Le piattaforme supportate sono: Windows 95, 98, NT, 2000 e XP.

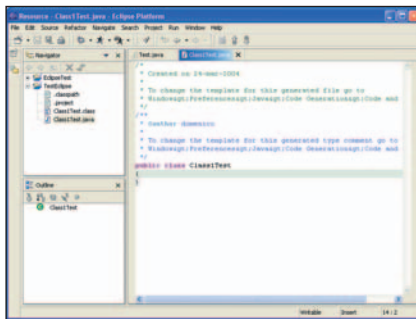
DEVLIB 1.1

Tutto l'occorrente per creare applicazioni multimediali.

Framework object-oriented realizzato completamente in C++ con l'obiettivo di fornire agli sviluppatori uno strumento potente e

sofisticato per la creazione di prodotti multimediali: giochi, screensaver, applicazioni grafiche (2D/3D), player audio e video, etc.
DevLib è completamente compatibile con gli ambienti di sviluppo Bloodshed's DevCpp e Microsoft's Visual C++ 2003, inoltre, utilizza le librerie *DevIL, FreeType 2, libjpeg, libmpeg2, libpng, TinyXML, unzip, ZLib, SDL, DirectX 9, FMOD* e *STL*.

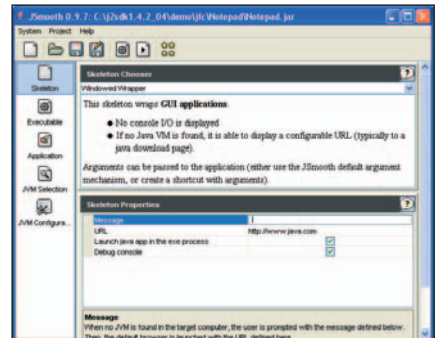
ECLIPSE 2.1.3

L'ultima release stabile del framework Open Source per lo sviluppo di applicazioni.

L'obiettivo della piattaforma Eclipse è quello di fornire ai programmatori un ambiente di sviluppo integrato (IDE) multipiattaforma per la realizzazione di qualsiasi tipo di applicazione non legato all'utilizzo di uno specifico linguaggio di programmazione.

La struttura del framework è estendibile, basata su un sistema di API e plugin che consente agli sviluppatori di estendere le funzionalità dell'ambiente di sviluppo per meglio adattarlo alle proprie esigenze.

JSMOOTH 0.9.7

Creare eseguibili a partire da file JAR

Tool Open Source per creare eseguibili standard per Windows (.exe) a partire da file Java .jar.
Il programma, definito Java Executable Wrapper, include nell'eseguibile creato le impostazioni normalmente settate da batch e avvia la JVM solo nel caso in cui questa sia presente nel sistema sul quale viene eseguito.

In caso contrario, avvisa l'utente e apre una connessione Internet per effettuare il download della JVM. Grazie a JSmooth non sarà più necessario distribuire le applicazioni Java complete di JRE, con un notevole guadagno in termini di spazio e semplicità di utilizzo per gli utenti dell'applicazione da distribuire.

MONO 0.30

Implementazione free del framework Microsoft .NET.

Il progetto Mono è nato da un' iniziativa Ximian (oggi è appoggiato anche da Novell) con l'obbiettivo di creare una versione Open Source per Unix della piattaforma di sviluppo Microsoft .NET, in maniera tale da permettere agli sviluppatori Unix di realizzare applicazioni .NET cross-plattform.
Mono include un compilatore per il linguaggio C#, un ambiente runtime chiamato *Common Language Runtime* (CLR) per il *Common Language Infrastructure* (CLI), supporto per ADO.NET e ASP.NET e numerose librerie di classi completamente compatibili con .NET.
Gli sviluppi futuri del progetto prevedono tra le altre cose la realizzazione di compilatori per il linguaggio di programmazione Visual Basic.NET già in fase di test, che sarà chiamato MonoBasic, e per il linguaggio di scripting Jscript .NET.

Macromedia RoboHelp X5

Creare Help professionali con un ambiente di sviluppo completo.

Un tool di altissimo livello professionale per la creazione di help sia per applicazioni desktop che Web Based. RoboHelp è in grado di creare sistemi di help utilizzabili attraverso un qualsiasi browser e su qualunque piattaforma. Le facilitazioni per lo sviluppatore sono molte e vanno dalla creazione in automatico di indici e glossari, alla generazione oggetti grafici, riducendo così i tempi di apprendimento del nuovo strumento.

UN SISTEMA UNIVERSALE

Qualsiasi sia il formato dell'help che vogliamo produrre, RoboHelp ci viene in contro grazie alla vastissima gamma di compatibilità che offre: Microsoft HTML Help, FlashHelp, WinHelp, WebHelp, JavaHelp, Oracle Help. Una volta popolato di contenuti, basterà un unico clic per ottenere l'help nel formato che desideriamo. RoboHelp può essere utilizzato proficuamente anche per la realizzazione di materiale cartaceo, grazie alla possibilità di avere in output documenti Word o PDF.

Fig. 1: ***Davvero completa l'interfaccia di RoboHelp.***

SVILUPPARE CON FACILITÀ

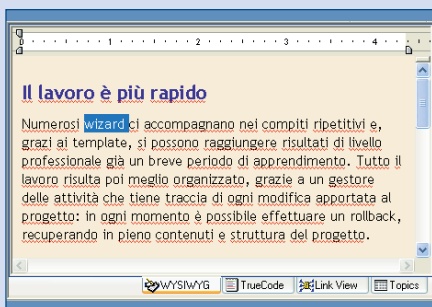
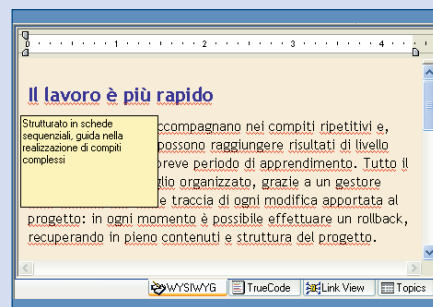
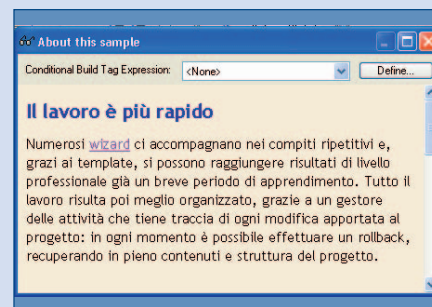
I nuovi utenti saranno potranno prendere confidenza con RoboHelp in modo molto graduale: per poter inserire i contenuti dell'help, oltre al potente editor integrato, è possibile utilizzare qualsiasi editor HTML o addirittura MS Word, delle attività che tiene traccia di ogni modifica apportata al progetto: in ogni momento è possibile effettuare un rollback, recuperando in pieno contenuti e struttura del progetto. La gestione dei contenuti risulta particolarmente efficace anche nelle situazioni di teamworking: ogni sviluppatore può accedere indipendentemente a specifiche porzioni del progetto, senza "pestare i piedi" agli altri componenti del team. Versione di valutazione valida quindici giorni.

IL LAVORO È PIÙ RAPIDO

Numerosi wizard ci accompagnano nei compiti ripetitivi e, grazie ai template, si possono raggiungere risultati di livello professionale già un breve periodo di apprendimento. Tutto il lavoro risulta poi meglio organizzato, grazie a un gestore

Fig. 2: ***I tool disponibili sono numerosi e tutti semplici da utilizzare.***

☑ RoboHelp X5
Produttore: Macromedia, Inc.
Sul Web: www.macromedia.it
Prezzo: € 779,00
Nel CD: *robohelp.exe*

AGGIUNGIAMO UNA NOTA DI POP-UP

1 Dopo aver ciccato sul tab *WYSYWYG*, selezioniamo la frase che dovrà essere linkata al pop-up testuale.

2 Dal menu *Insert*, selzioniamo la voce *Text-Only Popup* e inseriamo il testo che dovrà apparire. Il testo selezionato ora è blu e sottolineato.

3 Con un clic sull'icona degli occhiali (*View Selected Item*), possiamo apprezzare il lavoro finito. Un clic sul testo per vedere il Pop-up.

LEIF 2.0

Aumenta il potenziale delle applicazioni C++ con i Web Services.

Un framework che consente di creare Web Services in C++ o di trasformare, rapidamente, le nostre vecchie applicazioni in servizi Web. Basta fornire un file WSDL e LEIF si occuperà di generare tutto lo scheletro del servizio necessario a gestire tutti i dettagli di protocollo e di networking. Un ruolo fondamentale è giocato dai Wizard che guidano lo sviluppatore in tutte le fasi più delicate inerenti la creazione di un servizio. Sfruttando la potenza dei dati in XML e dei Web Services, LEIF permette di integrare rapidamente le applicazioni sviluppate in C++ all'interno delle piattaforme .NET e J2EE, sia interne all'azienda che provenienti da partner esterni o dai clienti. LEIF è un Framework leggero, flessibile e di tipo cross-platform che si basa sulle tecnologie standard quali l'HTTP, SOAP, WSDL e l'XML. Grazie all'utilizzo degli standard aperti e delle più collaudate tecnologie, LEIF minimizza i rischi di adozione e semplifica le attività di sviluppo. Consente inoltre di integrare senza problemi le applicazioni esistenti, garantendo la riduzione dei costi di sviluppo e le future evoluzioni, al passo con le esigenze dell'azienda. Versione di valutazione valida trenta giorni, richiede che sia installato Windows 2000/XP e Visual C++ 6.0.

Fig. 1: **Il processo che porta alla realizzazione di un nuovo servizio.**

☑ LEIF 2.0
Produttore: Quovadx, Inc.
Sul Web: www.roguewave.com
Prezzo: $1.495
Nel CD: ***leif200eval_win.zip***

Hackman Hex Editor 8.0 Lite

Per editare qualsiasi file con la facilità di un word processor

A dispetto del nome, questo prodotto non è dedicato esclusivamente al mondo hacker, ma a chiunque sia curioso di indagare la struttura dei programmi eseguibili. HackMan può infatti riportare in assembler qualsiasi eseguibile adatto ai Pentium. Benché la versione lite sia limitata nella dimensione dei file che può trattare, Hackman resta uno dei migliori disassemblatori reperibili al momento.
HackMan è anche un ottimo editor esadecimale e offre la capacità di criptare e decrittare file con un algoritmo a 128 bit. Tra le caratteristiche più interessanti annoveriamo il disk editor, la toolbar configurabile, un'interfaccia più veloce, il supporto per disassemblare codice Pentium 4 e molto altro ancora.
Con la semplicità di un Word Processor, sarà possibile leggere e modificare qualsiasi file, disco e qualsiasi zona della RAM. Molto utile la possibilità di controllare qualsiasi azione direttamente da un'apposita riga di comando, oltre che attraverso l'interfaccia grafica. E' possibile arricchire di nuove funzionalità l'ambiente, grazie al supporto per plug-in che possono essere realizzati con l'apposito SDK presente nel pacchetto di installazione.
La versione 8.0 è caratterizzata da un notevole incremento delle prestazioni (+200%) e numerose nuove funzionalità nell'editing dei file. Versione lite, è possibile gestire unicamente file la cui dimensione è inferiore ai 200KB. La licenza della versione Standard ha un costo di $24,99.

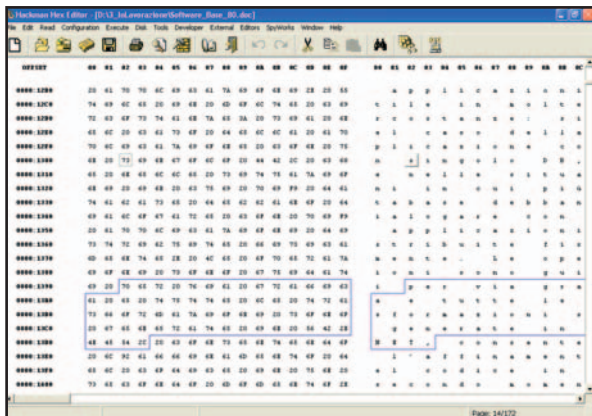



Fig. 1: **La lettura esadecimale di un file presente su disco.**

☑ Hackman Hex Editor 8.0 Lite
Produttore: TechnoLogismiki
Sul Web: www.technologismiki.com
Prezzo: gratuito per la versione Lite, $24,99 per la versione Standard
Nel CD: ***hack80.zip***

DeZign for Databases 3.2.1

Un tool per la modellazione dei database.

Davvero comodo questo tool che consente di modellare qualsiasi database, attraverso un diagramma entità relazione. Una volta pronto lo schema, DeZign può generare il database corrispondente in pressoché qualsiasi database esistente: Oracle, Interbase, IBM DB2, Sybase, MS Access, MS SQL Server, MySQL, PostgreSQL, Paradox, dBase, DBISAM, Advantage DB, FoxPro, Informix, NexusDB e Pervasive. Tra le funzionalità più interessanti, annoveriamo la possibilità di visualizzare il modello a diversi livelli di dettaglio, semplificando così le varie fasi di modellazione. Interessante l'introduzione del concetto di dominio (domain), che consente di specificare una struttura dati definita dall'utente. Tutte le informazioni che fanno riferimento ad un dominio possono dunque essere modificate in un'unica volta. Molti i miglioramenti introdotti in questa nuova release: uno tra tutti, la possibilità (con l'opzione "Propagate on Delete") di scegliere se la chiave esterna deve essere cancellata nel momento in cui viene a mancare il riferimento esterno. Tutti i progettisti apprezzeranno le capacità di report che, potendo essere prodotti a diverso livello di dettaglio, possono essere letti da tutte le parti coinvolte nel progetto. Versione di prova valida trenta giorni.

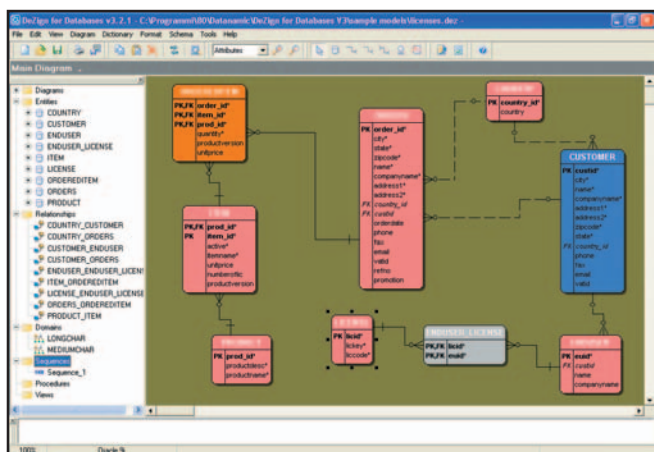



Fig. 1: *La splendida interfaccia, organizzata completamente in Wizard.*

☑ **DeZign for Databases 3.2.1**
Produttore: Datanamic
Sul Web: www.datanamic.com
Prezzo: $229
Nel CD: ***dezign.zip***

Astrum InstallWizard 2.02.1

Crea il tuo setup con una procedura semplice e guidata.

Astrum InstallWizard è un versatile software per la creazione di pacchetti di installazione. Potente e ben ideato, consente di arrivare al pacchetto di installazione completo attraverso una semplice e chiara procedura guidata. L'ottimo help e i numerosi tutorial aiutano anche i meno esperti a realizzare in pochi istanti pacchetti di livello professionale. Non rinuncia alla completezza comprendendo il supporto per file JPEG ed MP3. E' possibile utilizzare variabili utente, dividere I file di installazione su più dischi e interagire in vario modo con il registro di windows. Semplice ed altamente personalizzabile. Da rimarcare la presenza di un apposito Wizard per la creazione di update. Tutto il processo di installazione può essere finemente tarato sulle esigenze della nostra applicazione e, anche l'aspetto può essere ampiamente personalizzato. Il supporto dei temi (come i temi di Windows) consente un'ampia scelta, date un'occhiata al link qui riportato: *www.thraexsoftware.com/aiw/themes.html*
Questa release gestisce pacchetti di installazione la cui dimensione può arrivare fino a 4GB.

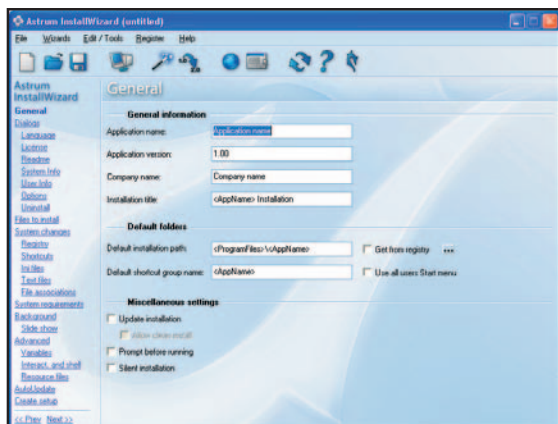



Fig. 1: *La splendida interfaccia, organizzata completamente in forma di Wizard.*

☑ **Astrum InstallWizard 2.02.1**
Produttore: Thraex Software
Sul Web: www.thraexsoftware.com
Prezzo: $49
Nel CD: ***aiw.exe***

EmEditor Professional 4.04

Un editor pronto a qualsiasi linguaggio

Un versatile editor testuale che, grazie al supporto per i plug-in può crescere insieme alle nostre esigenze. Molti sviluppatori l'hanno già scelto grazie soprattutto all'ampio supporto fornito per tutti i più diffusi linguaggi di programmazione: ASP, C++, C#, CSS, HTML, Java, JavaScript, JSP, Pascal (Delphi), Perl, PHP, Python, Ruby, SQL, Tex (LaTeX), VBScript, e Windows Script. Ognuno di questi linguaggi gode di un proprio syntax highlighting che semplifica la lettura di codice scritto da terze parte e riduce la possibilità di commettere errori durante la digitazione di nuovo codice. Versione di prova valida trenta giorni.

emed4epx.exe

Visual Integration Studio 2.5.2

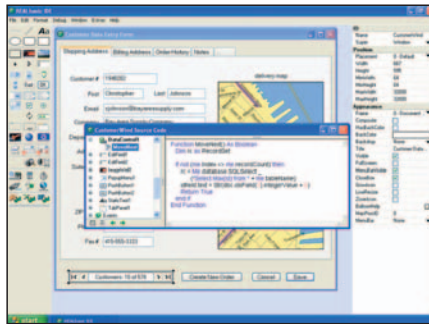
Estrae e trasforma dati da qualsiasi fonte

Un utile strumento visuale per Visual Basic .NET che consente di semplificare il processo di integrazione fra fonti di dati e applicazioni. Utile in molte circostanze: sia nel caso della applicazione con un singolo DB, che nelle situazioni in cui più database debbano dialogare con più applicazioni distribuite fisicamente. Le operazioni sono guidati per via grafica e tutte le trasformazioni sono generate in VB.NET, consentendo l'affinamento del codice in un secondo momento. Versione di valutazione valida trenta giorni.

VIShrNr.exe

Realbasic 5.5.1

Crea e compila applicazioni per Windows, Mac e Linux

Un ambiente di programmazione che rende disponibile anche ai meno esperti la possibilità di sviluppare applicazioni in pochissimo tempo, grazie anche alla ricca documentazione, ai numerosi tutorial e agli esempi inclusi. Con l'aggiunta del compilatore Linux, sarà possibile installare le applicazioni sviluppate con RealBasic anche su distribuzioni SuSE e Red Hat Enetrprise e, in generale, su tutte le distribuzioni Linux dotate di GTK+ 2.0 e di librerie CUPS. Quste piattaforme si vanno ad aggiungere a Windows, per Mac OS 8, Mac OS 9 e Mac OS X. Importanti i passi avanti fatti nella direzione di un miglior supporto per XML, Web Services ed il protocollo SOAP, insieme ad una migliorata compatibilità con i documenti generati dal Microsoft Office, con cui è ora possibile una reale sinergia. Senza contare che la sintassi è stata "limata" per farla aderire maggiormente a VBA, da cui differisce ora solo marginalmente. Di eccezionale utilità risulta la nuova possibilità di effettuare il debug delle applicazioni da remoto. Supportando ambienti "promiscui", è possibile lanciare il debug di un'applicaizone su Windows, effettuare il rebuild e lanciare direttamente l'applicazione su un Mac. Ad ogni breakpoint, saremo avvisati sulla machina Windows.

Gli utenti del Visual Basic di Microsoft non si troveranno disorientati, grazie ad una esplicita aderenza di Realbasic alle consuetudini della casa di Redmond nelle interfacce, senza contare che la migrazione per gli utenti Visual Basic è facilitata anche da un apposita utilità denominata VB Project Converter. Chi lavora un team potrà beneficiare del Project Manager che consente a più utenti di collaborare sullo stesso progetto. Versione dimostrativa valida dieci giorni. Al primo avvio è necessario cliccare su "Get a demo Key" per ottenere una chiave valida.

REALbasicSetup.exe

Sirid 1.18

Per gestire i tuoi progetti software

Un sistema professionale per la gestione di progetti spftware: con Sirid è possibile gestire il ciclo di vita di un'applicazione dal punto di vista dello sviluppatore. E' possibile assegnare compiti e priorità ai componenti del team ed i bug riportati, oltre ad essere memorizzati in un database, possono essere automaticamente inoltrati al responsabile del progetto. Grafici statistici e timesheet aiutano ad ottimizzare il lavoro degli sviluppatori. Sirid è accessibile attraverso un comune browser, non è dunque necessario alcuna installazione lato client e risulta garantita la delocalizzazione della forza lavoro.

sirid.exe

Tarma Installer 2.72.1521

Per avere pacchetti di installazione in più lingue

Attraverso una interfaccia particolarmente user-friendly, creare pacchetti di installazione con Tarma Installer è un vero gioco! Supporta tutte le piattaforme Windows (95, 98, Me, NT 4, 2000 e XP) e si segnala per numerose caratteristiche positive: piccoli pacchetti di installazione, interfaccia semplice e rapida da utilizzare, funzioni di installazione e disinstallazione intelligenti. E' possibile distribuire programmi, documenti, controlli ActiveX, font TrueType e OpenType, driver per periferiche, aggiornamenti di registro e molto altro ancora. Presenta funzionalità specifiche per la distribuzione sicura delle applicazioni sia via Internet si attraverso CD-ROM. Gratuito.

tin2.exe

Dataviewer Pro 3.5

Per gestire più database

Una comodissima plancia di comando per gestire parallelamente più database: Oracle, SQL Sever, Sybase e MySQL. La gestione di più database contemporaneamente è aiutata da un intelligente utilizzo dei colori: ad ogni connessione è associato un colore e tutte le finestre che fanno riferimento a quella connessione conservano il medesimo colore. Il tool consente confronti fra tabelle, esportazione di dati fra database e tutta una serie di funzioni che aiutano la gestione quotidiana e straordinaria dei DB. Versione di prova valida trenta giorni.

dvp_install.exe

Retroweaver 1.0 rc

Usa adesso il linguaggio del futuro

Se sei uno sviluppatore Java e sei curioso di provare le nuove caratteristiche della versione 1.5, questa è l'occasione giusta: Retroweaver è un generatore di bytecode che compila applicazioni e classi scritte con la nuova sintassi in modo da essere compatibili con la macchina virtuale 1.4. Potrete dunque sperimentare i generics, il polimorfismo parametrico e tutte le altre novità dell'1.5, senza dover cambiare il vostro ambiente di sviluppo.

retroweaver-all.jar

COMPONENTI .NET

FileCrypto 1.0.2

Crittografia facile!

Un componente per gli sviluppatori .NET che consente alle applicazioni di utilizzare tutte le capacità crittografiche del framework con grande semplicità. Crittare e decrittare diviene semplicissimo con tutti i principali algoritmi: DES, RC2, Rijndael, TripleDES. Versione dimostrativa, risultano disabilitate alcune funzioni.

fc10.zip

GPS.NET Visualization Tools 1.2

Metti un GPS nella tua applicazione

Un componente in grado di interpretare le comunicazione dei GPS integrabili con il PC. Integrare le funzioni di posizionamento nelle nostre applicazioni diventerà un gioco da ragazzi: basterà seguire gli esempi inclusi nel pacchetto di installazione. Versione di valutazione valida trenta giorni.

GPSVT110.zip

DemoWare 1.0

Distribuisci le demo delle tue applicazioni

Per la serie "Facciamoci pagare!", un componente che consente di creare versioni dimostrative e trial delle applicazioni che vogliamo distribuire. Semplicissimo da utilizzare: è sufficiente aggiungere il componente alla main form della nostra applicazione. Gratuito, richiede il .NET Framework 1.1.

DemoWare.zip

DataGridColumns .NET assembly 1.8.5

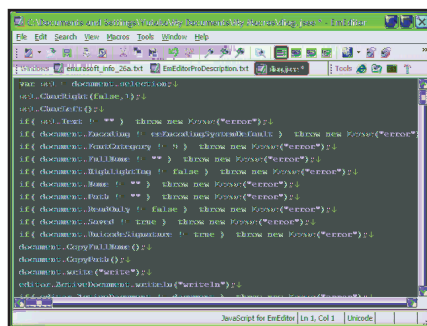
Un datagrid per movimentare l'interfaccia

Un pacchetto che consente di ridisegnare l'interfaccia delle nostre applicazioni .NET arricchendo i datagrid con la possibilità di utilizzare comboBox, pulsanti in stile XP, campi memo, controlli numerici e molti altri ancora.

DataGridColumnsTrial.zip

EzCrypto .NET 1.0

Crittare file, stringhe e stream

Un componente che consente di aggiungere ad applicazioni VB.NET e C# la possibilità di crittare file, stringhe e stream. Scritta completamente in C# (100% managed code), supporta tutti i più diffusi algoritmi, compresi RC2, Triple DES e Blowfish. Versione dimostrativa.

ezctsetup.exe

Dali 1.0

Una magia per connettersi ai database!

Dali rappresenta uno layer autoconfigurante che si pone fra l'applicazione ed i database: analizzando la struttura delle nostre classi e dei database che vogliamo interrogare, Dali genera automaticamente tutto l'SQL necessario alle operazioni di lettura, creazione e cancellazione su DB. Sarà possibile dialogare con i database senza conoscere SQL o ADO.NET. Versione di prova valida sessanta giorni.

Dali_1.0.0.0t.msi

.NET Communication Library 1.0

Iscrivi le tue applicazioni ai newsgroup!

Il .NET Framework soffre di alcune carenze nel campo dei protocolli: in particolare risulta assente il supporto al protocollo NNTP (Network News Transfer Protocol), necessario per interfacciarsi ai newsgroup della rete Usenet. Questo componente va a colmare questa lacuna, consentendo la realizzazione di applicazioni e siti ASP.NET in grado di interfacciarsi con i newsgroup. Gratuito.

CSpot_NETCommunicationLibrary.exe

LIBRERIE C++

Oxygen Mobile ActiveX Control 3.0

Controlla i telefonini Nokia

Un ActiveX che consente di gestire tutte le funzioni di qualsiasi telefonino Nokia: SMS, Java, giochi, gallerie, loghi, suonerie e altro ancora. Oxygen Mobile può anche leggere il codice IMEI, il livello della batteria, livello del segnale, versione dell'hardware e data e versione del firmware. Versione dimostrativa, risultano disabilitate alcune funzioni.

OxygenMobileActiveX.zip

Matrix ActiveX Component 3.1

Matrici senza segreti

Un componente che semplifica l'utilizzo dell'algebra matriciale in qualsiasi applicazione. Sono supportate tutte le operazioni: addizione, sottrazione, moltiplicazione, inversione, trasposizione e determinante. È anche possibile effettuare decomposizioni LU e decomposizioni di Colesky, calcolare autovettori e autovalori, lavorare con numeri reali e complessi, e molto altro ancora. Versione dimostrativa limitata a lavorare con matrici di dimensioni 6x6.

maxc31.zip

Packet Sniffer SDK 2.0

Per creare applicazioni di rete a basso livello

Una suite di componenti che consente di interagire a basso livello con i pacchetti di dati che viaggiano sulle reti. Molto efficace, risulta abbastanza complessa da utilizzare. Molto utili risultano essere gli oltre 4MB di codice sorgente esemplificativo inclusi nel pacchetto di installazione.

Versione di prova valida quindici giorni.

pssdk.zip

NCTAudioStudio ActiveX DLLs 2.4.3

Audio: controllo totale!

Una suite comprendente ben 17 controlli ActiveX per la manipolazione di dati di tipo audio: è possibile leggere e scrivere file, effettuare diversi tipi di conversione ed è presente anche una utile funzione di merge tra più file.

Il pacchetto può essere provato gratuitamente ma, se utilizzato a fini commerciali è necessaria la registrazione a pagamento.

In questo aggiornamento è stata aggiunto il supporto per il formato Mobile Voice. Questa nuova versione aggiunge il supporto per numerosissimi formati audio: WAV non compresso, WAV compresso (GSM, ADPCM, DSP, ALAW, ULAW, ALF2 CD e altri), MP3, MP2, VOX, RAW, WMA, AVI (audio), Ogg, Vorbis e altri ancora.

NCTAudioStudio2.exe

C#: proteggiamo le nostre applicazioni

L'attivazione del software

L'utilizzo non autorizzato del nostro software può rappresentare una perdita economica non indifferente. In questo articolo impareremo a tutelarci rendendo inutilizzabile il software non acquistato.

Il fenomeno dell'uso di software non regolarmente acquistato, affiancato all'under licensing (l'utilizzo di applicazioni su un numero di PC superiore al numero di licenze acquistate), è una delle principali cause di mancato guadagno per molte aziende. Premesso che nessun sistema è inviolabile (basta fare una ricerca su internet per accorgersene), in questo articolo impareremo a difendere i nostri prodotti da un uso improprio, scoraggiando il più possibile chi tenterà di utilizzarli al di fuori delle installazioni autorizzate. Nel corso degli anni abbiamo assistito al proliferare di sistemi di protezione del software: si va dalla chiave Hardware (di cui trovate tutti i dettagli a pagina 29), alla protezione con codici seriali fino ad arrivare alle procedure di attivazione. Il sistema più semplice prevede un unico codice seriale che il nostro software deve controllare per poter funzionare. Tale sistema è molto semplice da implementare, ma offre un livello di protezione piuttosto basso. Il primo sistema che proponiamo questo mese si basa sempre su un codice seriale ma, in questo caso, il codice è univoco, cioè generato da noi e solo dopo che il software è stato installato dal cliente. In questo modo, sebbene il cliente sia in possesso di tale codice, esso funzionerà solo sulla macchina in possesso dei parametri che abbiamo usato per il calcolo. Aggiungendo qualche altro piccolo accorgimento al codice del nostro prodotto, possiamo rendere la vita abbastanza difficile a chi cercherà di bypassare la protezione.

Fig. 1: *Diagramma di flusso della procedura di attivazione*

LE PRIME RIGHE DI CODICE

Il principio che sta alla base del nostro sistema di attivazione è che il nostro prodotto deve funzionare su un solo PC. Per far questo, dobbiamo vincolare il nostro software al PC su cui viene installato. A tale scopo, abbiamo la necessità di realizzare la porzione di codice che verrà inglobata nel nostro prodotto e il tool per generare il codice di attivazione (che deve restare a noi!). Nel codice allegato sono presenti tutti i commenti per meglio comprendere la logica nel suo insieme. Per prima cosa, analizziamo il flusso che la procedura di attivazione dovrà seguire per verificare che il software sia stato correttamente attivato. Come possiamo vedere dalla Fig. 1, abbiamo la necessità di avere due parametri da confrontare. Il primo sarà un parametro caratteristico del PC su cui il software è stato installato. A tale scopo, è stato scelto il numero seriale dell'Hard Disk. Tale codice è scritto direttamente dal produttore e non può essere modificato (a differenza del numero seriale del Volume). Il secondo è invece il codice di attivazione opportunamente archiviato nel PC del cliente. La sua assenza è sintomo che il software non è stato ancora registrato. Per recuperare il seriale dell'Hard Disk, ci viene incontro *Windows Management Instumentation* (WMI). Questo componente, incluso nelle versioni di Windows dalla 98 in poi, ci consente di accedere a tutta una serie di informazioni riguardanti l'hardware ed il software presente sul PC, compreso quello che a noi interessa. Analizziamo direttamente lo stralcio di codice contenuto nella classe *HardDisk.cs* presente nel codice allegato:

```
internal string GetSerial(){
ManagementObjectSearcher searcher = new
    ManagementObjectSearcher("SELECT SerialNumber
                    FROM Win32_PhysicalMedia");
```

```
    object so;String sn = null;
    foreach(ManagementObject wmi_HD in
                                       searcher.Get()) {
      if ( (so = wmi_HD["SerialNumber"]) != null && so
                                              is String ){
        sn =  ((string)so).Trim();
        break; }
    }
    return sn;
}
```

ManagementObjectSearcher non fa altro che recuperare una serie di informazioni relative al PC. La si può utilizzare tanto per recuperare informazioni sull'Hardware, quanto per il recupero di informazioni sui processi attivi, alle connessioni di rete ecc.
L'istanza di questa classe accetta come input una Query WMI che nel nostro caso consiste in

```
SELECT SerialNumber FROM Win32_PhysicalMedia
```

Win32_PhysicalMedia contiene i dati relativi alle periferiche di archiviazioni quali Hard Disk, CD Rom, registratori a nastro, dischi USB ecc.
Abbiamo dunque risolto il primo problema: il codice estratto sarà quello che il cliente dovrà fornirci in modo che il nostro tool possa calcolare il codice di attivazione. Tornando al diagramma di flusso di Figura 1, ci manca appunto il codice di attivazione.

CRITTOGRAFIA: LA CHIAVE DI AVVIO

Il sistema di base che utilizzeremo in questo programma sarà quello di crittografare il seriale dell'Hard Disk del cliente. In questo modo avremo un codice di attivazione che dipende direttamente tanto dal seriale, quanto dalle chiavi e dall'algoritmo che utilizzeremo per generare il codice. In un secondo momento, una volta compreso il meccanismo, potremmo anche divertirci a "sporcare" il seriale per renderne ancora più complessa l'operazione inversa. Fortunatamente, il .net Framework ci viene incontro fornendo già le implementazioni dei più diffusi algoritmi di crittografia. Tali algoritmi si suddividono in diverse famiglie tra cui distinguiamo quelli a chiave simmetrica e quelli a chiave asimmetrica (definiti anche a chiave pubblica). In un algoritmo a chiave simmetrica, sia il mittente che il destinatario utilizzano la stessa chiave per crittografare il testo in chiaro e decrittografare quello crittografato. Questo sistema è stato uno dei più diffusi ma è affetto da un problema: la trasmissione della chiave, che deve necessariamente avvenire in modalità sicura. Per risolvere il problema della trasmissione della chiave, sono nati gli algoritmi a chiave asimmetrica (detti anche a chiave pubblica). Il loro punto di forza è che per le operazioni di crittografia vengono utilizzate due chiavi distinte:

1. una pubblica, che quindi può essere fornita a chiunque, il cui scopo è quello di crittografare un testo in chiaro.

2. una privata, che deve essere necessariamente custodita segretamente in quanto, solo attraverso essa, è possibile decrittare un testo crittografato con la corrispondente chiave pubblica.

In questo modo è risolto il problema della trasmissione della chiave ma, gli algoritmi a chiave asimmetrica sono piuttosto lenti. Un buon compromesso è quello di utilizzare gli algoritmi a chiave simmetrica per crittografare grosse quantità di dati, e di utilizzare l'algoritmo a chiave asimmetrica per lo scambio della chiave. Nel nostro caso, per comodità implementativa, abbiamo utilizzato l'algoritmo a chiave simmetrica. Nel Framework .NET sono disponibili quattro classi che implementano tale algoritmo, tra queste abbiamo scelto quella che implementa l'algoritmo di *Rijndael* (*RijndaelManaged*), riconosciuto come uno dei più "resistenti" della sua famiglia. La trattazione teorica di questo algoritmo esula dallo scopo di questo articolo ma, per chi volesse approfondire, c'è un interessante sito da cui scaricare tutta la documentazione relativa, compreso un piccolo applicativo che spiega passo per passo come funziona l'algoritmo (vedi box a lato).

PASSIAMO ALLA PRATICA

Creiamo la classe che avrà il compito di crittografare e decrittografare le nostre stringhe. Tale classe ha il nome di *Cryptography* e implementa due soli metodi:

```
public string Crypt(string chiaro)
```

e

```
public string DeCrypt(string crittato)
```

che hanno l'evidente scopo di svolgere il lavoro che a noi interessa. Di particolare interesse, nella classe *Cryptography.cs*, è da notare:

```
byte[] key = {....};
byte[] IV= {....};
```

Questi due array di byte svolgono un ruolo fondamentale nella gestione della crittografia. L'array *key* rappresenta la chiave con cui la stringa in chiaro verrà crittografata. *IV* invece è l'*Initialization Vector*

L'algoritmo di *Rijndael* fu inventato da Joan Daemen e Vincent Rijmen a seguito di un concorso lanciato dal NITS *(National Institute of Standards and Technology)*. Lo scopo era quello di sostituire il precedente algoritmo, il DES, che iniziava ad essere insicuro grazie all'aumento della potenza di calcolo degli elaboratori. Tutte le informazioni su tale algoritmo sono reperibili sul sito http://www.esat.kuleuven.ac.be/~rijmen/rijndael/

(Vettore di Inizializzazione). Per comprendere meglio il suo utilizzo apriamo una piccola parentesi: per crittografare una stringa, normalmente, si utilizza sia la chiave che il risultato della codifica dei byte precedenti. All'inizio però, non abbiamo questo valore e il Vettore di Inizializzazione serve proprio a fornircelo. A questo punto abbiamo a disposizione tutti gli elementi per generare il nostro codice di attivazione valido.
Al metodo *Crypt(…)* dobbiamo passare la stringa in chiaro (seriale dell'Hard Disk) che verrà processata come segue:

```
ASCIIEncoding textConverter = new ASCIIEncoding();
RijndaelManaged rm = new RijndaelManaged();
public string Crypt(string chiaro){
  ICryptoTransform encryptor = rm.CreateEncryptor(
                                              key, IV);
  MemoryStream msEncrypt = new MemoryStream();
  CryptoStream csEncrypt = new CryptoStream(
      msEncrypt, encryptor, CryptoStreamMode.Write);
  //Converte la stringa in un array di byte
  bToCrypt = textConverter.GetBytes(chiaro);
  //Scrive i dati nello stream
  csEncrypt.Write(bToCrypt, 0, bToCrypt.Length);
  csEncrypt.FlushFinalBlock();
  bCrypt = msEncrypt.ToArray();
```

Dalla stringa in chiaro passata al nostro metodo, abbiamo ottenuto un array di byte contenenti il nostro codice. Sebbene sia possibile recuperare direttamente il codice di attivazione con l'istruzione:

```
string codice = Convert.ToBase64String(bCrypt);
```

è preferibile recuperarne la rappresentazione esadecimale, byte per byte. Non facendolo, otterremo un codice di attivazione testuale ma con lettere maiuscole e minuscole, simboli ecc, che diventerebbe difficile da dettare al cliente, nonché poco elegante.
Cicliamo quindi nel nostro array convertendo ogni byte nel corrispettivo valore esadecimale

```
StringBuilder sb = new StringBuilder(bCrypt.Length*2);
for ( int i = 0; i < bCrypt.Length; i++ ){
  sb.Append(bCrypt [i].ToString("X2"));
}
return sb.ToString();
```

Non è difficile immaginare che questa classe (ed in modo particolare questo metodo), farà da base per la realizzazione del tool di generazione del codice.

LA PROCEDURA DI ATTIVAZIONE

Nei paragrafi precedenti abbiamo imparato a recuperare il codice seriale dell'Hard Disk ed a crittografarlo. Ora passiamo alla procedura di attivazione. Dobbiamo innanzitutto individuare un sistema per archiviare il codice di attivazione in modo tale che possa essere recuperato e controllato in qualsiasi momento. Anche in questo caso, abbiamo diversi sistemi a disposizione da utilizzare. Possiamo infatti memorizzarlo sotto forma di file di testo, XML, binario ecc. Quello che ci conviene fare però, è riuscire a nasconderlo agli occhi dell'utente finale, non tanto per una questione di sicurezza (il codice è utilizzabile solo sul PC su cui è stato calcolato), ma per evitare che possa essere involontariamente eliminato o modificato. La scelta fatta è stata quella di memorizzare il suddetto codice all'interno del registro di Windows. Quest'ultimo, infatti, ci dà la possibilità di conservarlo in un luogo abbastanza sicuro, vista l'enorme quantità di dati presenti. Anche qui, il Framework .NET, ci fornisce gli strumenti adatti per svolgere le nostre operazioni in modo semplice e veloce. Il NameSpace *Microsoft.Win32* ci fornisce infatti tutte le classi per la gestione del registro di sistema e per la gestione dei messaggi generati dal sistema stesso.
Procediamo quindi all'archiviazione del codice di attivazione memorizzandolo in una sottochiave denominata *ioProgrammo\Attivazione*

```
RegistryKey rk;
......
rk = Registry.LocalMachine.OpenSubKey("Software",true);
rk = rk.CreateSubKey(@"ioProgrammo\Attivazione");
rk.SetValue("App",CodiceAttivazione);
```

La prima volta che eseguiamo il programma, tale sottochiave non esisterà nel registro del nostro cliente quindi, dobbiamo crearla.
La coppia di istruzioni che ci consente di farlo sono:

```
rk = Registry.LocalMachine.OpenSubKey("Software",true);
```

Innanzitutto apriamo il nodo *LocalMachine* e la sottochiave *Software*, sicuramente presenti nel registro. Con il valore booleano settato a *true*, ci predisponiamo alla scrittura.

```
rk = rk.CreateSubKey(@"ioProgrammo\Attivazione");
```

CreateSubKey ci consente ora di creare una nuova sottochiave denominata ioProgrammo che conterrà, al suo interno, una ulteriore sottochiave chiamata Attivazione. In questo modo otterremo una archiviazione strutturata delle informazioni.
A questo punto non ci resta che memorizzare il valore del codice di attivazione in un elemento che chiameremo *App*:

```
rk.SetValue("App",CodiceAttivazione);
```

NOTA

Nel codice allegato è presente un tool per la generazione della chiave e dell'IV da utilizzare nella classe *Cryptography.cs* al posto di quelli presenti. Avviare il programma *KeyGenerator.exe* e copiare i codici sia nel tool di attivazione che nel programma di attivazione.

SUL WEB

Informazioni più dettagliate sul WMI sono reperibili su MSDN, più precisamente su http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wmisdk/wmi/wmi_start_page.asp

Tale valore ci perverrà dalla *textBox* presente nel form di attivazione del prodotto, che il cliente compilerà con il codice che andremo a fornirgli.

REALIZZIAMO IL LOADER

Siamo alle battute finali della nostra procedura di attivazione. Abbiamo il tool generazione del codice ed abbiamo memorizzato il codice di attivazione nel registro ora… proteggiamo il nostro programma!
Riprendendo il diagramma di flusso di Fig. 1, dobbiamo fare in modo che il codice di attivazione memorizzato nel registro venga decrittografato e comparato con il seriale dell'Hard Disk. Se questo confronto ha esito positivo, possiamo avere la certezza che il software sia stato regolarmente attivato.
La classe *loader.cs* contiene un metodo denominato *chkAttivazione()*, il cui valore di ritorno è di tipo booleano. Tale metodo effettua esattamente il controllo che ci interessa:

```
private static bool chkAttivazione(){
  HardDisk hd = new HardDisk();
  Cryptography c = new Cryptography();
  string Seriale = hd.GetSerial();
  try{
RegistryKey rk = Registry.LocalMachine.OpenSubKey(
        @"Software\ioProgrammo\Attivazione\",false);
string StoredSN = (string)rk.GetValue("App");
    if ( c.DeCrypt(StoredSN) != Seriale ){
      return false;
    }else{
      return true; }
  } catch {
    return false;} }
```

Dopo aver recuperato il seriale dell'Hard Disk, il metodo legge il valore del codice di attivazione dal registro di Windows. Passandolo come valore al metodo *Decryt(…)* della classe *Cryptography.cs*, il codice di attivazione viene decrittografato in modo da poter essere comparato con il seriale dell'Hard Disk.
In tale metodo è importante notare una cosa: il codice che noi gli passiamo, sotto forma di stringa, in realtà è la rappresentazione esadecimale di ogni singolo byte del seriale crittografato. Per poterlo decrittografare, dobbiamo prima riconvertirlo da esadecimale ad array di byte.
A farlo se ne occupa il metodo *HexToByte*:

```
byte HexToByte( char hexchar ){
  byte b = (byte)hexchar;
  if ( b < 65 ) return (byte)(b - 48);
  else return (byte)((b % 32) + 9); }
```

Effettuiamo quindi il controllo tra il codice decrittato e il seriale dell'Hard Disk:

```
if ( c.DeCrypt(StoredSN) != Seriale )
{
  return false;
}
else
{
  return true;
}
```

Il valore booleano della classe *chkAttivazione()*, viene restituito al metodo chiamante (il *main*):

```
public static void Main() {
  Application.EnableVisualStyles();
  if (chkAttivazione()){
    Application.Run(new Conferma());
  }else{
    Application.Run(new MascheraAttivazione()); }
}
```

Se restituisce *true*, allora carichiamo il form principale del nostro programma (quello che conteneva il metodo *main*). Se torna *false* invece richiediamo la maschera di attivazione (Fig. 2).

Fig. 2: ***Il form con cui richiediamo il codice di attivazione.***

CONCLUSIONI

La sicurezza di questo sistema dipende molto da come gestiamo la gestione del codice di attivazione. Sporcare il seriale dell'Hard Disk, calcolare il codice su più seriali ricavati da Hardware diverso, utilizzare sistemi di crittografia a chiave asimmetrica ed introdurre ulteriori controlli nel codice del nostro software, non faranno altro che incrementare il livello di protezione e complicare la vita agli utenti… più smaliziati. Sul prossimo numero vedremo inoltre come consentire al cliente di ricevere il codice seriale attraverso un web service.

Michele Locuratolo

Una volta attivato il software, apparirà una maschera di conferma. Se riapriamo il file eseguibile ed abbiamo fatto tutto correttamente, accederemo direttamente alla suddetta maschera. Per effettuare altri tentativi, dobbiamo rimuovere la chiave di registro in cui abbiamo memorizzato il codice di attivazione. Possiamo farlo o andando direttamente nel registro, o usando l'apposito pulsante nella maschera di conferma.

Costruire e controllare una chiave di protezione

Creare e programmare una chiave hardware

La protezione dei componenti hardware e software è di fondamentale importanza nella sicurezza dei sistemi informatici. Scopriamo come realizzare una chiave Hardware riconfigurabile e riprogrammabile.

La sicurezza dei sistemi informatici si è rivelato un problema di importanza cruciale fin dall'avvento dei primi Personal Computer. La protezione dei propri sistemi è divenuta, con il passare degli anni una delle principali preoccupazioni di chiunque possegga un elaboratore elettronico. È ormai opinione diffusa che qualunque programma possa essere in qualche maniera manomesso e che possano essere aggirate le barriere di protezione che sono state predisposte per limitarne l'utilizzo al solo personale autorizzato. Fino ad ora abbiamo accennato alle preoccupazioni dell'utente generico, ma dal punto di vista del programmatore il pensiero di proteggere i propri programmi dalla copiatura indiscriminata, oppure semplicemente di garantire ai propri utenti che chi utilizza un certo prodotto sia semplicemente il legittimo proprietario porta alla definizione di meccanismi di protezione generalmente di tipo software. Uno dei metodi più frequente violazione dei sistemi di sicurezza è la decompilazione del codice e la successiva ricerca di metodi di 'bypass' della protezione: le biblioteche ormai traboccano di testi che trattano ogni tipo di tecnica riguardanti le metodologie più recenti di violazione dei sistemi di sicurezza. In questo articolo desideriamo presentare un sistema di protezione hardware chiamato comunemente 'chiave hardware', strutturato in modo tale da permettere la realizzazione del maggior numero possibile di strategie di protezione software per espanderne al massimo le potenzialità e la resistenza ad eventuali tentativi di violazione. Il sistema sarà strutturato in modo tale da presentare capacità 'all pourpose' in modo da potere essere utilizzato sia per la protezione di software applicativi, che di componenti hardware per mezzo di appositi driver, che ne gestiscano il funzionamento vero e proprio. La caratteristica più sorprendente che vogliamo fornire alla nostra chiave hardware è di essere riprogrammabile: quale sistema di sicurezza può essere più difficile da violare di uno che cambia continuamente le proprie strategie di protezione? Si noti che nell'ultima frase si è parlato non per caso di 'strategie' e non di 'strategia': infatti il nostro sistema potrà implementare più di un tipo di protezione, utilizzando differenti al-

Fig. 1: L'apparecchiatura PC Explorer light è prodotta e commercializzata dalla Elisys s.r.l. e può essere acquistata sul Web all'indirizzo www.pcexplorer.it.

Fig. 2: L'immagine mostra una vista di insieme dei cablaggi della chiave hardware.

Nel CD allegato o sul Web è disponibile un filmato che mostra la chiave hardware in funzione. *(Chiave_Hardware.AVI)*

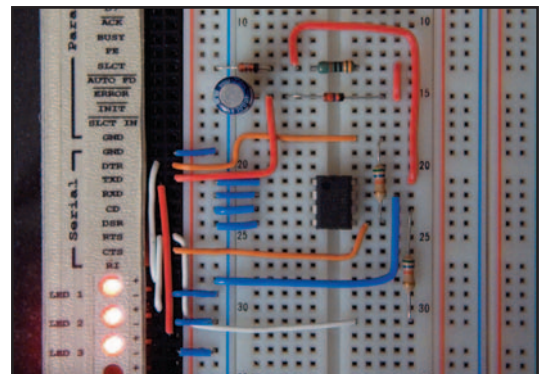



Fig. 3: Premendo il pulsante 'Hardware key ON' sul software di controllo tutte le linee della chiave hardware vengono poste a livello logico 'High', fornendo alimentazione al circuito.

goritmi e metodologie che si attivano in modo non facilmente prevedibile per garantire la sicurezza del software o dell'hardware che si intende proteggere.

PERCHÉ UNA CHIAVE HARDWARE

Una delle note leggi sulla sicurezza dei sistemi sostiene che la 'sicurezza dal lato client' non esiste: precisiamo che come 'lato client' intendiamo la parte del sistema che è a diretto controllo dell'utente, comprendendo in senso più ampio anche la parte relativa ai sistemi non connessi in rete.

Fig. 4: ***Il software descritto in queste pagine è presente nel CD allegato alla rivista completo di codice sorgente è altresì presente sul Web. Lo schema elettrico del circuito di controllo lo trovate nel file "Chiave_Hardware_Schema_Elettrico.bmp".***

Questa affermazione, molto generale peraltro, si applica perfettamente al nostro caso, poiché la parte del sistema sotto controllo dell'utente può essere analizzata e modificata da questi in modo diretto. Qualora l'utente abbia mezzi, conoscenze e tempo a sufficienza può teoricamente aggirare qualunque metodo di protezione predisposto sulla propria macchina. Quanto espresso ci porta alla conclusione che per garantire una maggiore sicurezza, occorre che il dispositivo di protezione venga predisposto al di fuori della parte del sistema che è sotto l'assoluto controllo dell'utente.
Per essere difficilmente attaccabile, deve potere adattare e mutare le proprie caratteristiche in modo da sopperire ad eventuali violazioni che dovessero verificarsi: in poche parole deve potere essere riprogrammata. Una ragionevole sicurezza di un sistema può essere ottenuta quindi utilizzando una chiave hardware che abbia la caratteristica di essere riprogrammabile.

N 1 EEPROM PCF85116-3
oppure 24C16
N2 Res. 5600 Ohm _ Watt
N1 Res. 330 Ohm _ Watt
N1 Zener 5,1V _ Watt
N1 Diodo 1N4148
N1 Condensatore
100uF 16V

I componenti sono reperibili presso il sito www.pcexplorer.it

ANALISI DELLO SCHEMA ELETTRICO

Il progetto hardware del sistema è incentrato sul componente PCF85116-3, EEPROM *('Electrically Erasable Programmable Read Only Memory)* con ha una capacità di memoria di 16 Kbit organizzati in otto blocchi da 256x8 bit. Per maggiori informazioni su questo componente si rimanda il lettore all'articolo dello stesso autore pubblicato su questa rivista nel numero precedente. Lo schema elettrico è riportato di seguito ed è inoltre incluso nel CD allegato alla rivista, per maggiore comodità del lettore nel file: *Spunto_HardwareKey_Programmer.zip*: il circuito è depositato e quindi protetto dalle leggi in vigore sull'argomento. Analizzando lo schema, si nota innanzi tutto la memoria EEPROM sul lato sinistro del progetto, collegata tramite le due resistenze di *'pull up' R1* e *R2* alla porta seriale e precisamente alle linee *RTS* e *DTR*. Lo stato logico di queste linee (rappresentante anche la situazione presente sul BUS I2C), può essere monitorizzato per mezzo dei LED *DL1* e *DL2*. Le procedure di gestione del BUS in questione sono state descritte nel dettaglio nell'articolo precedente e vengono illustrate sinteticamente nel paragrafo relativo all'implementazione software in C++. La linea di ingresso della porta seriale *CTS* è deputata alla lettura della memoria, secondo il protocollo I2C. Nella parte alta dello schema si nota la connessione alla linea TXD, normalmente utilizzata per l'invio di dati seriali, che nel nostro caso invece ha lo scopo di fornire l'alimentazione alla chiave hardware, senza limitare l'utilizzo della porta seriale per la trasmissione dati: la chiave è quindi trasparente nei confronti di programmi che non sono interessati al suo utilizzo. L'alimentazione del circuito avviene per mezzo del diodo *D1* che ha lo scopo di 'separare' la linea *TXD* della porta seriale dalla chiave hardware ed allo stesso tempo di consentire il passaggio di corrente soltanto quando sulla linea è presente una tensione positiva (si ricordi che a seconda del tipo di driver della porta seriale possono essere presenti valori di tensione fino a +/-25Volt).
Il gruppo di componenti *C1-R3-DZ1* infine, provvede a stabilizzare la tensione per garantire la corretta alimentazione della chiave hardware.

LA REALIZZAZIONE DEL CIRCUITO

La lista dei componenti necessari viene riportata a lato di queste pagine e nel circuito elettrico, per comodità e su richiesta dei lettori all'interno del file incluso al CD con il nome *Chiave_Hardware_Schema_Elettrico.bmp*. Sul lato destro dello schema, si possono notare le connessioni alle linee relative alla porta seriale e di alimentazione dell'apparecchiatu-

ra PC *Explorer light*. Durante la realizzazione del circuito si raccomanda di operare la massima attenzione affinchè nella manipolazione della memoria EEPROM non ne vengano toccati i terminali metallici, per evitare eventuali danneggiamenti dovuti a cariche elettrostatiche. Il cablaggio è stato realizzato utilizzando l'apparecchiatura *PCExplorer light*, in alternativa è possibile utilizzare le tecniche costruttive convenzionali, ovvero dotandosi di stagno, saldatore ed una buona dose di pazienza. Il lettore ha in ogni caso tutte le informazioni necessarie alla realizzazione della parte hardware, e più avanti troverà il software di gestione, completo di componenti pronti all'uso, del programma compilato e funzionante dotato di codice sorgente.

CONTROLLO E PROGRAMMAZIONE

Il software proposto in queste pagine è in grado di provvedere alla programmazione della chiave hardware, leggerne il suo contenuto e verificare, per mezzo di un banale algoritmo, una chiave di protezione. Il codice sorgente, (scritto in C++) viene messo a completa disposizione del lettore ed è disponibile nel CD con il nome. Il programma gestisce lo scambio di informazioni tra PC e chiave hard-ware quindi, dal momento che il passaggio di informazioni è bidirezionale, per convenzione chiameremo *Transmitter* il dispositivo che invia i dati e *Receiver* quello che li riceve. Master sarà il dispositivo di controllo e Slave i sistemi controllati.

```
//-----------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "SpuntoHardwareKeyProgrammerUnit.h"
//-----------------------------------------------------------
#pragma package(smart_init)
#pragma link "SpuntoLedComponent"
#pragma link "TSpuntoHardwarePortIO_unit"
```

Dall'intestazione del programma si nota che vengono utilizzati due componenti, dei quali vengono forniti i rispettivi file per la corretta installazione: in particolare *SpuntoLedComponent* permette la gestione dei LED e *TSpuntoHardwarePortIO_unit* consente lo scambio di informazioni a basso livello con la porta seriale. La funzione che segue inizializza la porta seriale in termini di parametri di comunicazione e provvede ad alimentare il circuito elettronico, inviando il valore *0xFF* sulla porta e di conseguenza sulla linea di trasmissione *TXD* (si veda il paragrafo relativa al progetto elettronico).

```
//-----------------------------------------------------------
void TSpuntoHardwareKeyProgrammerForm::
HardwareDeviceSerialON(unsigned short
                    HardwareDeviceComPort)
{ SpuntoHardwarePort->WritePort(HardwareDeviceComPort
                    +1,0x00);  //INT OFF
...
SpuntoHardwarePort->WritePort(HardwareDeviceComPort,
                    0xFF); //SENDS FF TO COM PORT}
//-----------------------------------------------------------
```

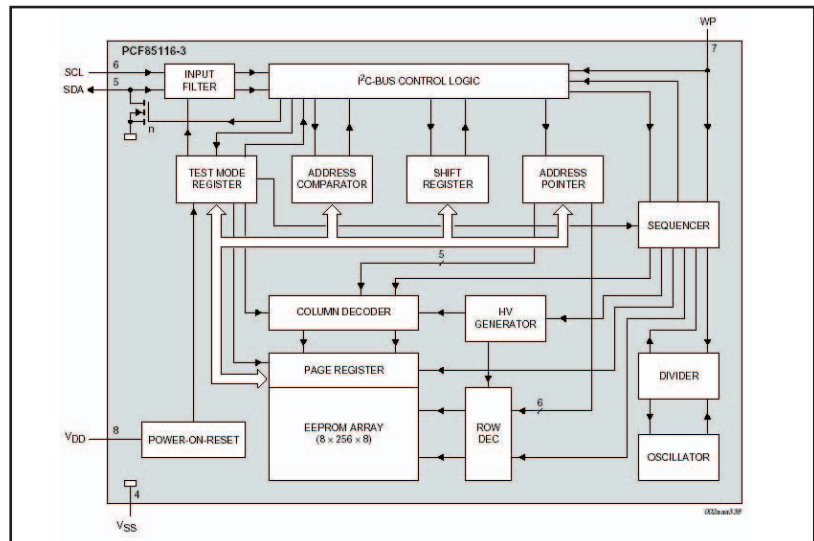



Fig. 5: ***Nell'immagine di figura si riporta lo schema logico e la piedinatura del circuito integrato PCF85116-3, reperibili in forma completa su Internet all'indirizzo: http://www.components.philips.com/ (cortesia Philips Semiconductors).***

La funzione *Write_Data* consente la scrittura della memoria EEPROM secondo il protocollo *I2C*. Riassumendo brevemente quanto viene ad accadere, possiamo dire che ogni byte è seguito da un bit di Acknowledge (ACKN), il quale rappresenta un livello High posto sul bus dal Transmitter, con un relativo impulso sulla linea di clock *SCL*: lo Slave è obbligato ad inviare un *Acknowledge* dopo la ricezione di ciascun byte. Il *Master Receiver* deve generare un *Acknowledge* dopo la ricezione di ciascun byte ricevuto dallo *Slave Transmitter*. Il dispositivo che genera una condizione di *Acknowledge* deve porre *SDA* a livello logico *Low* in modo stabile durante il periodo di clock a livello High. Al bit di *Acknowledge,segue* l'indirizzo del byte che si intende memorizzare ed un ulteriore *Acknowledge* a cui fa seguito finalmente il dato da registrare in memoria. A questo punto si può procedere alla scrittura sequenziale di dati (fino a 32 consecutivi), alternando ciascun byte con una sequenza di *Acknowledge*. È possibile terminare la scrittura in qualunque momento facendo seguire alla sequenza di bit di dati una condizione di mancato *Acknowledge* ed una condizione di *STOP*, rendendo così possibile la scrittura anche di un solo byte.

```
//-----------------------------------------------------------
void TSpuntoHardwareKeyProgrammerForm::
```

ACQUISTARE PC EXPLORER LIGHT

L'apparecchiatura PC Explorer light è prodotta e commercializzata dalla Elisys s.r.l. e può essere acquistata sul web all'indirizzo www.pcexplorer.it oppure inviando una e-mail all'indirizzo pcexplorer@elisys.it, od anche telefonicamente al numero 0823/468565 o via Fax al: 0823/495483.

```
                         Write_data(int Address, byte Data)
{ //
byte Control_Byte, Memory_Block;
  Memory_Block=Address >>8;
  Memory_Block=Memory_Block <<1;
  Control_Byte=0xa0; //1010 0000 Control Code 1010
  Control_Byte=(Control_Byte | Memory_Block);
                                 //Sets the control byte
  Start();  //Start bit
  Output_byte(Control_Byte);   //Control Byte
  Acknowledge();
  Output_byte(Address&0xff); //Address Byte
  Acknowledge();
  Output_byte(Data);          //Data Byte
  Acknowledge();      //Stop bit
  Stop(); }
//-----------------------------------------------------------
```

Fig. 6: *L'immagine mostra il risultato della scrittura della cella di indirizzo '10', contenente il valore '25'.*

La funzione di lettura della memoria (*Read_Data*) opera con un procedimento analogo alla scrittura fino all'invio dell'indirizzo del dato da leggere. Impostato l'indirizzo del byte che deve essere letto ed il relativo *Acknowledge* si ha una seconda condizione di *START* ed un nuovo preambolo, questa volta con il bit *R/W* posto a '1' (Lettura). Dopo la condizione di Acknowledge si ha finalmente la lettura da parte del Master Transmitter, divenuto Master Receiver che preleva gli otto bit di dati dalla memoria EEPROM Slave Transmitter. La procedura di lettura anche in questo caso può essere ripetuta in modo sequenziale fino ad un massimo di 32 byte, oppure interrotta inviando una condizione di mancato *Acknowledge* ed una sequenza di STOP rendendo anche in questo caso possibile la lettura anche di un solo byte.

```
//-----------------------------------------------------------
byte TSpuntoHardwareKeyProgrammerForm::
                              Read_data(int Address)
{ //
byte Control_Byte, Memory_Block, DataValue;
  Memory_Block=Address >>8;
  Memory_Block=Memory_Block <<1;
  Control_Byte=0xa0; //1010 0000 Control Code 1010
  Control_Byte=(Control_Byte | Memory_Block);
                                 //Sets the control byte
  Start();
  Output_byte(Control_Byte);
  Acknowledge();
  Output_byte(Address&0xff);
  Acknowledge();
  Start();
  Control_Byte=(Control_Byte | 0x01);
                          //Sets R-/W bit to 1 (Read)
  Output_byte(Control_Byte);
  Acknowledge();
  DataValue=Input_byte();
  Stop();
  return(DataValue); }
//-----------------------------------------------------------
```

Alla pressione del tasto di programmazione della chiave, viene eseguita la funzione che segue, che provvede a recuperare i valori di indirizzo e dato da memorizzare dai relativi componenti *SpinEdit*, eseguendo alla scrittura e poi operare la verifica della correttezza del dato scritto. Una funzione analoga che omettiamo per brevità provvede alla lettura della memoria ed a verificare l'attendibilità del dato così ottenuto.

```
//-----------------------------------------------------------
void __fastcall TSpuntoHardwareKeyProgrammerForm::
                     WriteEEPROMClick(TObject *Sender)
{int ProgramAddress;
byte ProgramData, WrittenData, IsEEPROMReady;
 ProgramAddress=CSpinEditAddress->Value;
 ProgramData=CSpinEditData->Value;
 Write_data(ProgramAddress, ProgramData);
                                  //Writes the EEPROM
 WrittenData=Read_data(ProgramAddress);
                                 //Reads written data
 if ((WrittenData==ProgramData)& ( WrittenData!=0))
                               //Verifies written data
  { AckLabel->Caption="SUCCESSFUL";
    EEPROMStatusLabel->Caption="READY"; }
  else
  { AckLabel->Caption="NOT SUCCESSFUL";
    EEPROMStatusLabel->Caption="NOT PRESENT"; }}
//-----------------------------------------------------------
```

Di seguito viene riportato una semplice applicazione della chiave hardware: la funzione EEPROMCheck provvede a verificare che nella cella indirizzata dal valore contenuto nel componente *Spinedit KeyAddress* sia presente il valore contenuto in *KeyData:* se questa condizione è verificata, provvede a comunicare che il controllo è andato a buon fine (OK), altrimenti rappresenta una situazione di mancato superamento del test (FAULT). Il software di controllo è stato collaudato con Win 3.x, Win 9x e Win Me, se si utilizza Win 2000, XP oppure NT, per evitare l'errore di 'Privileged Instruction' generato da questi ultimi sistemi operativi quando si tenta di accedere alle porte hardware del PC, è possibile utilizzare il driver PortTalk (*PortTalk22.zip*), scaricabile all'indirizzo: *www.beyondlogic.org/porttalk/porttalk.htm*.

```
//----------------------------------------------------------
byte TSpuntoHardwareKeyProgrammerForm::
                    EEPROMCheck(void) //EEPROM Check
{ int  KeyAddress;
byte WrittenData, KeyData;
  KeyAddress=KeyAddressSpinEdit->Value;
  KeyData=KeyDataSpinEdit->Value;
  WrittenData=Read_data(KeyAddress);
                                    //Reads written data
  if (WrittenData==KeyData)      //Verifies written data
   { KeyOKLabel->Caption="OK"; //Verification passed
     AckLabel->Caption="SUCCESSFUL";
    KeyOKLabel->Font->Color=clGreen;
    EEPROMStatusLabel->Caption="READY";}
   else
   {
    KeyOKLabel->Caption="FAULT";
                              //Verification not passed
    AckLabel->Caption="NOT SUCCESSFUL";
    KeyOKLabel->Font->Color=clRed;
    EEPROMStatusLabel->Caption="NOT PRESENT"; }
   return(WrittenData); }
//----------------------------------------------------------
```

UTILIZZO DELLA CHIAVE HARDWARE

Per dimostrare le funzionalità di base della chiave hardware e collaudarne il funzionamento, proponiamo una semplice applicazione che ci permetta di verificare il valore di un singolo byte contenuto all'interno della memoria EEPROM che ha lo scopo di simulare una semplice password di accesso ad un programma. Prima di affrontare l'esempio, provvediamo a verificare un'ultima volta tutte le connessioni elettriche: completato il controllo, siamo pronti a collegare il circuito alla porta parallela del PC, ovviamente a computer rigorosamente spento, accendiamo il calcolatore e lanciamo il programma di controllo. A titolo di esempio, supponiamo di volere registrare la nostra password alla locazione di memoria *N 10* con il valore impostato a *25*.
Premiamo i pulsanti 'SDA High' e 'SCL High' e controlliamo che i LED posti sulla realizzazione hardware si accendano, come mostrato nella figura precedente: questo controllo ci permette di verificare che la porta seriale ed il software siano configurati correttamente.
Impostiamo i valori di 'Address' e 'Data' rispettivamente a *10* e *25* e premiamo 'Write EEPROM': dovremmo vedere i due LED lampeggiare velocemente, fatto che ci conferma l'avvenuta programmazione della memoria. Nel CD allegato alla rivista è disponibile un filmato che mostra la chiave hardware in funzione *(Chiave_Hardware.AVI)*. Premiamo a questo punto il pulsante 'Verify Key' dopo avere impostato i parametri relativi alla nostra password simulata ed avremo il risultato della verifica della sua correttezza (*OK* oppure *FAULT*). Ovviamente l'esempio che è stato proposto ha un puro valore didattico, nel prossimo articolo affronteremo alcune tecniche di protezione ben più avanzate utilizzando la nostra chiave hardware.

CONCLUSIONI

In queste pagine abbiamo sviluppato, analizzato e collaudato una chiave hardware di protezione per Personal Computer sotto forma sperimentale e di prototipo.

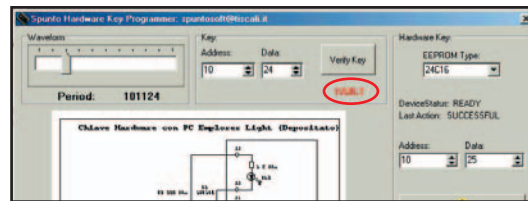



Fig. 7: ***Nel caso non vengano inseriti i parametri corretti di codifica, il programma non riconosce la validità della chiave hardware.***

Il progetto dello schema elettrico, tutti i collegamenti necessari, il software compilato ed i relativi codici sorgenti sono stati messi a completa disposizione del lettore: un filmato dimostrativo è disponibile sul CD ROM allegato alla rivista. Un doveroso e sentito ringraziamento è dovuto alla Philips Semiconductors per avere permesso la pubblicazione delle informazioni relative alla memoria EEPROM PCF85116-3. Il lettore vorrà comprendere che nonostante quanto esposto in queste pagine sia stato debitamente verificato e collaudato, tuttavia viene riportato a scopo illustrativo e di studio, pertanto l'editore e l'autore non sono da considerare responsabili per eventuali conseguenze derivanti dell'utilizzo di quanto esposto in questa sede, soprattutto per la tipologia e la complessità dell'argomento.

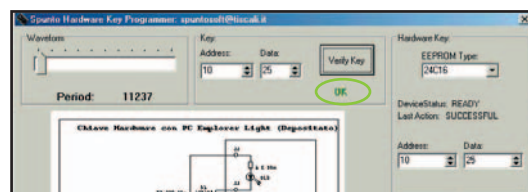



Fig. 8: ***La verifica della chiave di codifica ha successo inserendo i valori corretti di indirizzo e dato e premendo il tasto 'Verify key'.***

Nel prossimo appuntamento tratteremo alcune tecniche di protezione dei sistemi per mezzo della chiave hardware proposta in questa sede, come ad esempio la memorizzazione di password, il controllo dell'utilizzo di un programma e la protezione delle informazioni.

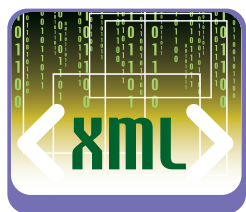
Luca Spuntoni

IL SOFTWARE

Il codice sorgente della applicazione e tutti i componenti necessari sono reperibili sul CD allegato e sul Web, all'interno del file: *Spunto_HardwareKey_Programmer.zip.* Il software di controllo è stato collaudato con Win 3.x, Win 9x e Win Me, se si utilizza Win 2000, XP oppure NT, è possibile utilizzare un driver, per evitare l'errore di *'Privileged Instruction'* quale *'PortTalk' (PortTalk22.zip)*, scaricabile del sito: http://www.beyondlogic.org

La trasformazione dei documenti

XSL in OFFICE 2003

Continuiamo l'esplorazione di Office 2003, descrivendo le trasformazioni XSL. Approfitteremo del contesto per analizzare anche la protezione e la formattazione dei documenti Word XML.

Nel precedente appuntamento abbiamo presentato le nuove caratteristiche della famiglia di applicazioni Microsoft Office 2003 evidenziando che il supporto nativo della tecnologia XML le ha rese più efficienti sia nella creazione di soluzioni desktop che nel caso di applicazioni distribuite. In particolare abbiamo visto che Excel e Word supportano strumenti evoluti come: schemi XML, trasformazioni XSL (Extensible Stylesheet Language) e Smart Document. Inoltre, abbiamo presentato un esempio che utilizza uno schema XML (file .XSD) e una trasformazione XSL (file .XSLT). L'applicazione d'esempio, che completeremo in questo appuntamento, è un sistema integrato di archiviazione per la gestione dei curriculum dei laureati. Il sistema prevede che i dati vengano inseriti attraverso un documento Word-XML, archiviati in dei documenti XML ed infine importati in un database Access. Nei curriculum, per semplicità, sono inseriti soltanto le informazioni basilari cioè: dati anagrafici, indirizzo, dati laurea ed i dati per catalogarli (data richiesta, numero protocollo e note). In questo appuntamento completeremo il sistema di archiviazione e parallelamente descriveremo altre caratteristiche di Office 2003, come: protezione e formattazione documenti Word-XML, utilizzo nelle macro VBA dei campi e degli oggetti XML, utilizzo delle trasformazioni XSL, ecc. Iniziamo descrivendo i nuovi strumenti di protezione offerti da Word.

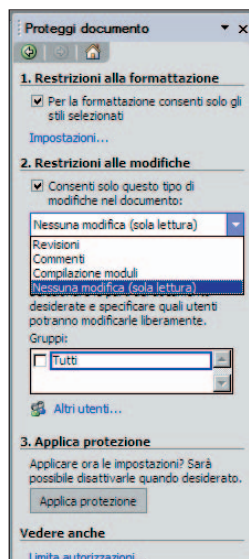



Fig. 1: *La scheda protezioni del riquadro attività.*

PROTEZIONE DEI DOCUMENTI

Nei documenti Word 2003 sono stati introdotti due gradi di protezione che permettono di controllare la formattazione ed il contenuto. Queste protezioni non necessariamente devono essere applicate a tutto il documento. Per esempio è possibile definire un documento con una formattazione mista, tag XML ed elementi Word, proteggere la formattazione ed il contenuto e lasciare modificabile lo spazio all'interno dei tag XML. Le protezioni possono essere gestite attraverso la scheda "proteggi documento" del riquadro attività. Questa scheda è divisa in due parti, una consente d'impostare le restrizioni alla formattazione, l'altra al contenuto. Inoltre, quando si selezionano le restrizioni al contenuto è possibile specificare le aree modificabili e il tipo di modifica consentita, da scegliere tra le seguenti: solo lettura, revisione, commenti e compilazione moduli (come mostrato in Fig. 1). Ora, come accennato, creiamo il documento Word-XML, che consente di inserire i dati dei curriculum, e su di esso impostiamo delle restrizioni che consentono di modificare soltanto il contenuto dei tag XML.

CURRICULUM WORD-XML

Create un nuovo documento WordML (nominato *curricula.xml*) ed associategli lo schema *curriculum.xsd*, definito nel precedente articolo. Con gli strumenti messi a disposizione da Word dividete il documento in almeno tre parti (anagrafica, indirizzo e laurea). A tal fine potete usare delle tabelle e delle celle colorate. In ogni parte del documento inserite gli appropriati tag XML come predisposto nello schema XSD (controllate la Fig. 2). Dopo aver inserito i tag, predisponete le protezioni alla formattazione e al contenuto. Questo lo potete fare selezionando l'interno dei tag XML e spuntando le caselle di controlli restrizione alla formattazione e alle modifiche (opzioni solo commenti e per tutti gli utenti) della scheda protezioni del riquadro attività e premendo il pulsante *applica protezione* (attenzione, selezionate l'interno di un tag, applicate le restrizioni e seguite le indicazioni fornite sul riquadro attività). Per rendere il tutto più interessante, prevedete dei campi (fields) nei tag XML del numero di protocollo e della data di presentazione (*datarichiesta*). In particolare, inserite un campo di tipo *data* e un campo numerico (di tipo *DocVariable*). Questi campi li potete inserire attraverso la finestra gestione campo che compare selezionando la voce *campo* del menu inserisci, oppure utilizzando del codice VBA come il seguente.

```
Selection.Fields.Add Range:=Selection.Range,
                    Type:=wdFieldEmpty, Text:= _
"DOCVARIABLE  protocollo ", PreserveFormatting:=True
Selection.Fields.Update
```

Il codice precedente permette d'inserire un campo numerico in un punto del documento ed aggiornarne il contenuto (*Update*).
Un modo alternativo per inserire i campi è quello di utilizzare il modello di oggetti Word XML. Esso, infatti, permette di "scorrere" l'albero che rappresenta il documento XML, con un'espressione *XPath*, e ricavare il punto del documento (cioè il nodo XML) dove inserire i campi.
Di seguito riportiamo la macro che seleziona il nodo curriculum/datarichiesta ed inserisce un *campo data*.

```
Private Sub Inscampo()
 Dim objElement As XMLNode
 Dim strElement As String
 Dim strPrefix As String
 strElement = "/x:curriculum/x:datarichiesta
'stringa Xpath per selezionare il nodo datarichiesta
 strPrefix = "xmlns:x=""" & ActiveDocument _
 .XMLSchemaReferences(1).NamespaceURI & """"
 Set objElement = ActiveDocument _
 .SelectSingleNode(strElement, strPrefix)
 objElement.Range.Select
 Selection.Fields.Add Range:=Selection.Range,
                    Type:=wdFieldEmpty, Text:= _
 "CREATEDATE  \@ ""dddd d MMMM yyyy"" ",
                         PreserveFormatting:=True
End Sub
```

Il metodo *SelectSingleNode* restituisce un oggetto *XMLNode* impostato con i dati di un nodo dell'albero XML, questo nodo è selezionato in base ad un'espressione *XPath*, la sintassi delle *SelectSingleNode* è la seguente:

```
SelectSingleNode(XPath, PrefixMapping,
                    FastSearchSkippingTextNodes)
```

PrefixMapping è facoltativo ed indica il namespace (o prefisso) usato per i nodi; *FastSearchSkippingTextNodes* (un boolean facoltativo) serve per ignorare o includere, durante la ricerca, i nodi text. Il valore predefinito è *False* cioè non ignorarli.
Ricordiamo che la *proprietàXMLSchemaRferences* restituisce l'insieme degli schemi associati ad un documento.
Modificando opportunamente l'espressione *XPath* è possibile selezionare e quindi impostare o cancellare qualsiasi nodo. Ora occupiamoci della trasformazione XSL che ci consente di salvare i dati del documento XML in un formato che non include sotto alberi.

TRASFORMAZIONI XSL

Una XSLT è un programma (contenuto in un file con estensione XSLT) che definisce come trasformare un documento XML in un altro documento XML o in un file di un altro formato (HTML, XHTML, SVG, …). Per esempio, il bilancio di un'azienda, impostato in un documento Word, attraverso delle XSLT può essere adattato a tutti i tipi di Browser e a tutti i tipi di schermi anche a quelli dei Pocket PC. Così se è necessario fare qualche cambiamento al bilancio, basta modificare il documento XML Master. Se invece si vuole cambiare la visualizzazione basta modificare le trasformazioni. In Word 2003 le XSLT possono essere associate al documento attraverso la finestra Opzioni XML o direttamente durante il salvataggio del documento. Descriviamo un primo esempio di file XSLT, rifacendoci all'XSD "persona" descritto nel precedente articolo.

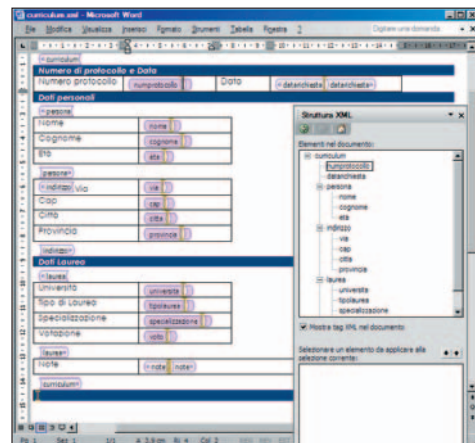
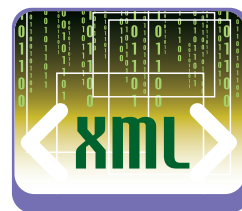



Fig. 2: *Il documento Word-XML con formattazione Word, Tag XML e protezioni.*

LA TRASFORMAZIONE DA XML A HTML

Per creare un file XSLT basta il blocco note, però se volete essere più produttivi vi consigliamo di utilizzare l'editor XML SPY (*www.xmlspy.com*) che, oltre a segnalare alcuni errori, vi aiuta nella fase di digitazione del codice. Il primo elemento di un file XSLT deve, necessariamente, essere il tag *<xsl:stylesheet>*, cioè:

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
…
</xsl:stylesheet>
```

Nel tag *<xsl:stylesheet>* bisogna inserire le regole di trasformazione, queste sono espresse attraverso gli elementi template. Per esempio, per convertire il file XML che contiene i dati di una persona in un file HTML possiamo usare la seguente trasformazione.

```
…
<xsl:template match="persona">
<html>
<body>
<p> <b>Cognome: </b>
<xsl:value-of select="cognome"/>
</p><p> <b>Nome: </b>
<xsl:value-of select="nome"/>
</p><p> <b>Eta: </b>
```

NOTA

FOR-EACH SELECT

Se in una trasformazione abbiamo una radice con più sotto nodi dello stesso tipo, per esempio il nodo *<persone>* che contiene più nodi *<persona>*, per selezionare dei sotto nodi, per esempio i cognomi, dobbiamo usare il costrutto *"for-each select"*, cioè

```
…
<xsl:template match=
                  "persone">
<xsl:for-each select=
                  "persona">
 <xsl:value-of select=
                  "cognome"/>
…
```

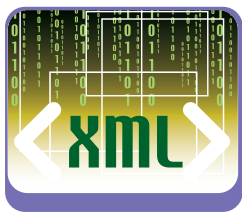



Fig. 3: I file XML salvati, con l'opzione solo dati, da WordML ed attraverso la nostra trasformazione.

```
<xsl:value-of select="eta"/>
</p>
</body>
</html>
</xsl:template>
...
```

Il template precedente permette di creare un file HTML a partire da un documento XML simile al seguente (discusso nel precedente appuntamento).

```
<?xml-stylesheet
 type="text/xsl" href="htmltpersona.xslt"?>
<persona>
<nome>Giuseppe</nome>
<cognome>Vatteneapesca</cognome>
<eta>56</eta>
</persona>
```

I tag *<? … ?>* delimitano il codice che consente, al processore di Internet Explorer, di individuare il file XSLT (*htmltpersona.xslt*). Per eseguire la trasformazione basta copiare i due pezzi di codice in due file del blocco note con estensione XSLT per il primo pezzo e XML per l'altro. I due file devono essere salvati nella stessa directory ed il file XML deve essere aperto con Internet Explorer. Nell'esempio si può constatare che una trasformazione XSL per HTML è un misto di codice HTML (notate i tag *<html>*, *<body>*, *<p>* *<b>* ecc.) e di espressioni XSLT. Inoltre, potete constatare che i template hanno un doppio ruolo: selezionare i nodi ed applicare le regole di trasformazione. Infatti, prima s'identifica il nodo del file XML sorgente (nel nostro caso *match="persona"*), di livello superiore, e poi si applicano le regole *"value_of select"* cioè prendi il valore all'interno dell'elemento.

NOTA

La proprietà *XMLSaveDataOnly* è un boolean che stabilisce se il documento sarà salvato con tutti i *Tag (=False)* o soltanto con quelli personalizzati *(=True)*. La proprietà *XMLUseXSLTWhenSaving*, invece, specifica se il documento verrà salvato attraverso una trasformazione XSL; se impostata su *True* deve essere utilizzata insieme alla *XMLSaveThroughXSLT* che permette di specificare il nome e il percorso della trasformazione.

LA XSLT PER I CURRICULUM

Dopo questa breve introduzione sull'XSLT, definiamo la trasformazione per i documenti XML del nostro sistema di archiviazione. Questa trasformazione, come accennato, serve per convertire dei documenti XML (che contengono i dati dei curriculum) in altri documenti XML più facilmente importabili in una tabella Access. A tal fine abbiamo pensato ad una trasformazione che elimini i rami dell'albero del documento XML, (cioè che porti tutti i nodi sotto la radice). Facciamo notare che questo è necessario perchè i documenti XML creati attraverso lo schema *curriculum.xsd* presentano tre rami – *persona, indirizzo* e *laurea* - e quando sono importati in Access (attraverso il menu carica dati esterni - da file XML) creano diverse tabelle: una principale ed una associata ad ogni ramo. Dunque, dobbiamo implementare una trasformazione che converte un documento XML in un documento XML con una diversa struttura ma che contiene gli stessi dati. Di seguito riportiamo un estratto della trasformazione, il file completo lo trovate nel CD allegato: *curriculum.xslt*.

```
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:vecchio="http://tempuri.org/curricula.xsd"
xmlns:nuovo="http://tempuri.org/curriculatab">
<xsl:template match="/vecchio:curriculum">
  <nuovo:curriculum>
    <xsl:apply-templates/>
  </nuovo:curriculum>
</xsl:template>
<xsl:template match="vecchio:numprotocollo">
  <nuovo:numprotocollo>
    <xsl:value-of select="."/>
  </nuovo:numprotocollo>
</xsl:template>
...
</xsl:template>
<xsl:template match="vecchio:note">
  <nuovo:note>
    <xsl:value-of select="."/>
  </nuovo:note>
</xsl:template>
</xsl:stylesheet>
```

Nella trasformazione, prima di tutto, abbiamo definito tre namespace uno per gli elementi XSL e gli altri per gli elementi dei due documenti XML: sorgente (namespace vecchio) e generato (namespace nuovo). Dopo la definizione degli spazi dei nomi, con un template selezioniamo il nodo radice (curriculum) e creiamo il *<nuovo:curriculum>* per il documento XML "generato". Nel template usiamo l'elemento *<xsl:apply-templates>*. Questa regola, al processore di file XSTL, indica che bisogna eseguire i template di tutti i nodi visibili da *<vecchio:curriculum>* ed i risultati, di queste esecuzioni, devono essere inseriti nel nodo che contiene *<xsl:apply-templates>*. Notate che con gli altri template si valutano i valori dei nodi del documento sorgente e si costruiscono i nodi del nuovo documento. Il codice precedente deve essere inserito in un file del blocco note nominato *tabcurriculum.xslt*. Descriveremo ora, come associare questo file ad un documento XML di Word.

USARE LE XSLT IN WORD

Una XSLT può essere associata ad un documento Word in vari modi, noi abbiamo scelto quello più semplice che si basa sull'uso del comando *salva con*

nome (Save As). Con questo comando, infatti, è possibile selezionare il formato XML ed attraverso un pulsante (nominato *Trasforma*) avviare la ricerca del file XSLT. Questo, naturalmente, può essere fatto anche attraverso delle righe di codice VBA. Per esempio possiamo prevedere del codice che salvi i curriculum in formato XML, attraverso la *tabcurriculum.xslt,* assegnandogli un nome univoco basato sul numero di protocollo (per esempio *"curriculum+numero di protocollo"*). Come capiremo tra poco, abbiamo previsto che questi file vengano salvati nella directory C:\Wordscheme\modelli e che la trasformazione sia contenuta nella directory *C:\Wordscheme*. Ora vediamo due macro, una per incrementare il protocollo e l'altra per salvare il documento. Queste naturalmente vanno utilizzate nel file Word-XML *curriculum.xml*.

```
Public Sub incprotocollo()
 On Error GoTo errore
  ActiveDocument.Variables("protocollo").Value = _
  ActiveDocument.Variables("protocollo").Value + 1
  ActiveDocument.Tables(1).Columns(2).Cells(1).Select
  Selection.Fields.Update
  ActiveDocument.Tables(1).Columns(4).Cells(1).Select
  Selection.Fields.Update
  Exit Sub
errore:
 ActiveDocument.Variables.Add Name:="protocollo",
                                        Value:="1"
 Resume Next
End Sub
Public Sub salvaconnome()
 incprotocollo
 With ActiveDocument
    .XMLSaveDataOnly = True
    .XMLUseXSLTWhenSaving = True
    .XMLSaveThroughXSLT =
                    "C:\wordscheme\tabcurriculum.xslt"
 End With
 com = ActiveDocument.Path + "\" + ActiveDocument.Name
 ActiveDocument.SaveAs FileName:=
                "c:\Wordscheme\modelli\curriculum" _
 + CStr(ActiveDocument.Variables("protocollo").Value)
 ActiveDocument.XMLSaveDataOnly = False
 ActiveDocument.XMLUseXSLTWhenSaving = False
 ActiveDocument.SaveAs com
End Sub
```

Notate che nella *incprotocollo* per selezionare i campi *datarichiesta* e *protocollo*, utilizziamo il fatto che questi sono inseriti in delle tabelle. Inoltre, notate che la prima volta che si cerca di incrementare il campo protocollo è generato un errore che viene catturato ed utilizzato per impostare il valore della variabile ad *1*. Per rendere il tutto più funzionale potete utilizzare la seguente macro che crea una nuova barra di menu.

```
Public Sub newmenu()
 Dim con As CommandBarButton
 Set newBar = CommandBars.Add(Name:="XmlBar", _
 Position:=msoBarTop, Temporary:=True)
 newBar.Visible = True
 Set con = newBar.Controls.Add(Type:=msoControlButton)
 con.FaceId = 0
 con.OnAction = "salvaconnome"
 con.Picture = LoadPicture(
                          "c:\Wordscheme\salvacon.bmp")
 `salvacon.bmp è l'immagine associata al nostro
                                 pulsante salvaconnome
End Sub
```

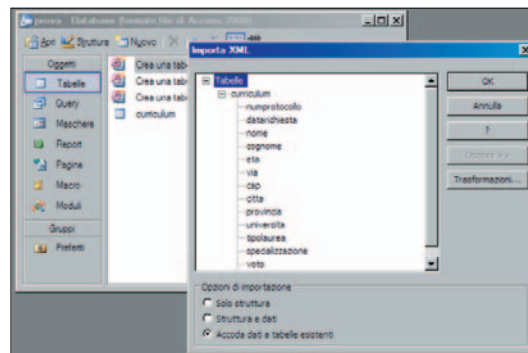



Fig. 4: **Access consente l'import di dati da file XML.**

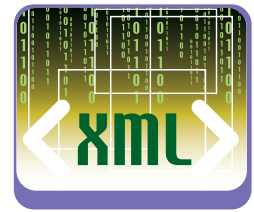
Le parti principali del sistema sono state definite, resta soltanto da stabilire come importare i dati nel database Access. Il metodo più semplice è quello di utilizzare, direttamente, le funzionalità di Access. Per esempio potete procedere nel seguente modo:

1. create un nuovo database Access;
2. selezionate la voce di menu file/caricadatiesterni/importa;
3. selezionate un file XML.

Ripetete i passi precedenti con l'opzione "accoda dati a tabella esistente".

CONCLUSIONI

Il prototipo, di sistema di archiviazione, che abbiamo presentato è servito per illustrare le nuove caratteristiche di Office 2003. Ora tocca a voi renderlo funzionale! Potrete, ad esempio: integrare i dati gestiti; prevedere, nella struttura XML, la gestione di più curriculum; implementare delle macro per impostare e cancellare il contenuto dei nodi XML; migliorare l'interazione con Access.
Buon lavoro!
Nel successivo appuntamento ci occuperemo dello Smart Document SDK per Office 20003 e creeremo una *Smart Fattura*, seguiteci!

Massimo Autiero

XSL

L'XSL nasce intorno al 1997 quando il W3C (www.w3.org) rilascia i primi documenti sulla filosofia generale del linguaggio. Attualmente l'XSL lo possiamo definire come una famiglia di raccomandazioni per la definizione di trasformazioni e presentazioni XML, essa è composta da tre parti: XSTL (XSL Transformations) che è un linguaggio per trasformazioni XML; XPath (XML Path Language) che sono delle espressioni usate per interrogare i documenti XML e XSL-FO (XSL Formatting Object) un vocabolario XML per specificare una semantica di formattazione per documenti XML.

VARIABILI

Nei documenti Word è possibile gestire delle variabili globali contenute nell'insieme *Variables* (che è gestito come una collezione). Esse sono invisibili all'utente e possono essere associate a dei campi (contenuti nell'insieme *fields*) *DocVariable*, a tal proposito controllate la procedura che permette d'incrementare il numero di protocollo.

Le API per la gestione delle email

Un filtro Antispam in Javamail

Progettare un sistema antispam in Java presenta alcune interessanti problematiche di progettazione orientata agli oggetti ed utilizzo di API standard della piattaforma J2EE, come Javamail. Scopriamone i segreti.

Fidanzate ucraine, prodotti per la pelle, servizi per la promozione del proprio sito Web. Ed ogni tanto qualche eminente esponente del governo o di società sudafricane, con a disposizione una cifra mai inferiore ai 20 milioni di dollari, si offrono di cederne una grossa parte, in cambio di non si sa bene cosa. Le nostre mail box sono piene di offerte, non richieste, per una varietà di prodotti e servizi. La cosidetta spam, o posta spazzatura, ormai è un problema globale che interessa milioni di persone; si calcola infatti che gran parte del traffico della Rete sia generato appunto da questa posta non desiderata. Come ogni fenomeno di globale, attorno allo spam si sta creando un'industria: chi si occupa di bulk mail, dell'invio cioè di mail in massa, il cui scopo prevalente è quello di raggiungere la maggior parte di destinatari; chi traffica indirizzi di posta di possibili acquirenti, offrendo interi database di nominativi "sicuri". Dall'altra parte, stanno anche nascendo strumenti per contrastare questo fenomeno, da quelli legislativi, che prevedono un maggior controllo sull'abuso della posta elettronica, da quelli informatici, che stanno portando alla costruzione di software di filtro, basati su tecnologie di diverso tipo.

COMBATTERE LO SPAM

Le tecniche per combattere lo spam sono diverse, ma si possono identificare tre tecnologie affermate. Alcuni client di posta, come *Mail* di Apple (Fig. 1), integrano un filtro bayesiano, basato cioè su ricerca statistica. Dopo un periodo di training, dove l'utente indica al programma quali sono messaggi indesiderati e quali no, è possibile passare in modalità automatica, dove il filtro cercherà di "indovinare" quali messaggi di posta siano non graditi, sulla base dell'esperienza accumulata durante la fase di training. Un'alternativa è quella di costruire una white

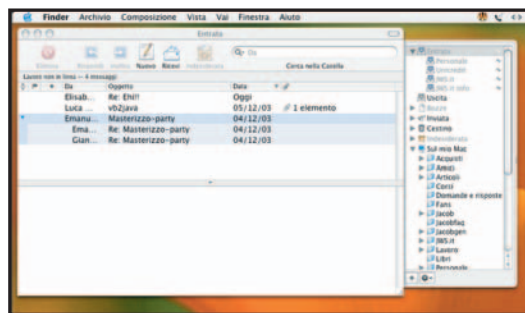



Fig. 1: Il software di Apple per la gestione delle e-mail include filtri bayesani contro la posta spazzatura.

list, una lista cioè di mittenti attendibili, gli unici i cui messaggi in ingresso verranno accettati dal sistema. La costruzione della lista può avvenire manualmente, oppure con un sistema automatico, come nel caso di Knowspam (*www.knowspam.net*).
Quando un mittente cerca di inviare un messaggio ad una casella di posta protetta da Knowspam, se il mittente è sconosciuto, il server gli invia un messaggio di posta chiedendogli di dimostrare che sia una vera persona e non, ad esempio, un programma di invio di bulk mail. La verifica viene fatta presentando una immagine gif che contiene un numero poco leggibile (riconoscibile dunque da un essere umano e non da un OCR) e chiedendone la digitazione. Se il numero corrisponde, il nominativo del mittente viene inserito nella white list. Una terza possibilità è quella di valutare il messaggio ricevuto alla ricerca di elementi peculiari della posta spazzatura, gli stessi elementi che fanno capire ad un osservatore umano che il messaggio è solo pubblicitario. Questi software, tra cui Spam Assassin (*http://spamassassin.org*) contengono una serie di regole che valutano il messaggio associando un punteggio, se questo supera cinque, il messaggio è considerato spazzatura. Alcuni elementi valutati sono, ad esempio, l'eccessivo uso dell'HTML, la presenza di testo nascosto, la presenza di url che contengono la parola "remove",

ma anche la presenza di testo di colore blu (spesso utilizzato nei messaggi spam) o la presenza di parole chiave relative al sesso.

COSTRUIAMO UN FILTRO

Progettare un sistema antispam in Java presenta alcune interessanti problematiche di progettazione orientata agli oggetti ed utilizzo di API standard della piattaforma J2EE, come Javamail. L'approccio scelto per implementare questo sistema di filtro è quello di realizzare una serie di regole di valutazione per determinare un punteggio da assegnare al messaggio, sulla falsariga di Spam Assassin. La caratteristica principale del sistema è quello di essere espandibile, di consentire cioè una facile aggiunta o rimozione di regole di valutazione, grazie alla dinamicità di Java ed all'uso delle interfacce. Il sistema può dunque essere considerato un framework, in quanto fornisce semplicemente il servizio di punteggio di un messaggio di posta. Sarà poi un programma specifico che si occuperà di utilizzare questo punteggio: potrebbe essere un modulo del server di posta, come Spam Assassin, oppure potrebbe essere integrato nel programma di posta elettronica dell'utente. Ad ogni modo, verrà realizzato anche un programma di verifica della mailbox, per mostrare in pratica l'uso del framework. La struttura generale del sistema prevede l'interfaccia *Rule*, che definisce il tipo che deve implementare ciascun singolo filtro, che sarà dunque contenuto in ciascuna classe che implementa l'interfaccia *Rule*. Questa prevede un solo metodo *rate()*, che si aspetta un messaggio di posta e ritorna un valore float che ne indica il punteggio. Per comodità vengono anche definite le costanti *NONE, ACCEPTED* e *REJECTED* che indicano rispettivamente: nessun punteggio, messaggio sicuramente accettato e messaggio sicuramente rifiutato. Il listato completo è il seguente:

```
package it.bigatti.antispam;
import javax.mail.*;
interface Rule {
static final float NONE = 0;
static final float ACCEPTED = -1;
static final float REJECTED = 999.0F;
float rate( Message message ) throws
                    MessagingException, RuleException; }
```

Ci possono essere infatti regole che non restituiscono un punteggio di probabilità, ma riescono a determinare immediatamente la validità o meno del messaggio; è il caso delle regole basate su white o black list. In questi casi, se un mittente è presente in white list (potrebbe ad esempio essere tratta dalla nostra rubrica indirizzi), questo è sicuramente accettato. Contrariamente, se il mittente è presente in black list, è già noto che questi è uno spammer.

ARCHITETTURA

Come sistema espandibile, si sarebbero potuto inserire centinaia di regole di valutazione ma, per ovvie ragioni, ci si è limitati ad includere le seguenti (i numeri tra parentesi sono il punteggio ottenuto per ciascuna regola, se questa è verificata):

- *white list (ACCEPTED);*
- *black list (REJECTED);*
- il messaggio ha priorità alta (0.5);
- il messaggio ha un link ad una pagina di rimozione (0.8);
- il mittente ha nel nome un underscore seguito da numeri (2.2);

Un diagramma delle classi semplificato è presente in Fig. 2.
Come si può notare, dall'interfaccia *Rule* deriva la classe astratta *AbstractRule*, che implementa alcuni servizi di utilità comune, tipicamente di elaborazione del messaggio da valutare. In particolare, sono contenuti i seguenti metodi:

```
//...
abstract class AbstractRule implements Rule {
public String getSender( Message message ) throws
                                   MessagingException {
//... }
public String getHtml( Message message )
throws MessagingException, IOException {
//... }
String getPartText( Part part ) throws
MessagingException, IOException {
//... }
}
```

Il metodo *getSender()* ritorna il mittente di un messaggio, *getHtml()* ritorna la parte html di un messaggio multipart; il metodo *getPartText()* ritorna il contenuto testuale di una specifica parte di un messaggio multipart. Il supporto ai messaggi multipart è indispensabile, in quanto oramai la maggior parte dei messaggi che vengono inviati dagli spammer hanno contenuti html organizzati secondo que-

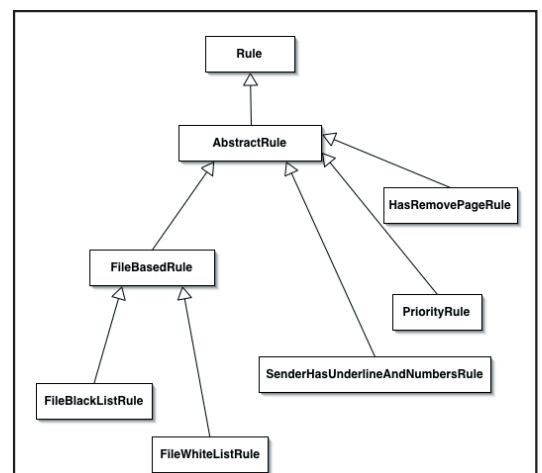



Fig. 2: ***Dall'interfaccia Rule derivano tutte le classi del framework antispam.***

sto formato. Da *AbstractRule* derivano le classi *HasRemovePageRule, SenderHasUnderlineAndNumbersRule, PriorityRule*, che implementano tre delle regole sopra elencate. Si noti come i nomi delle classi riprendono il tipo implementato *(Rule)* e che, anche a scapito di nomi piuttosto lunghi, cerchino di descrivere compiutamente il loro scopo. La scelta dei nomi delle classi è problematica soprattutto in un sistema dove le regole potrebbero crescere e diventare centinaia. In questo caso è necessario probabilmente suddividere le regole in diversi sottopackage, in modo da organizzarle per argomento (ad esempio un package per i controlli sul mittente, uno per quelli sulla priorità, uno per quelli sul contenuto, suddiviso a sua volta in quelli per contenuto html e quelli per il contenuto di semplice testo, e così via). Un'altra sottoclasse astratta di *AbstractRule* è anche *FileBasedRule*, che fornisce una base comune a tutte le regole che hanno a che fare con file di configurazione (nel nostro caso *white* e *black list*). Da *FileBasedRule* derivano infatti *FileBlackListRule* e *FileWhiteListRule;* si noti che questo ramo della gerarchia delle classi è studiato per operare, per semplicità, con file di testo. Un altro ipotetico ramo della gerarchia potrebbe trattare con liste remote, ad esempio accessibili in http. Si potrebbero dunque avere classi come *URLBasedRule, URLBlackListRule* e *URLWhiteListRule.*

UNA ALTERNATIVA DI DESIGN

Nella realizzazione di questo progetto, si è scelto di intraprendere una via nuova, costruendo una gerarchia di classi a sé stante. Una alternativa sarebbe potuta essere quella di basare le regole di valutazione dei messaggi sul sottosistema di ricerca di Javamail, che modella diverse condizioni di confronto per varie parti di un messaggio di posta, al fine di eseguire ricerche di messaggi che soddisfino determinati criteri. Sebbene si possano trovare similitudini tra le classi di ricerca di Javamail e le esigenze di un filtro antispam, la differenza concettuale impone, per chiarezza, la realizzazione di una gerarchia di classi autonoma.

IMPLEMENTARE LE REGOLE

L'implementazione di una regola basata su lista è semplice. Nel caso di *FileBlackListRule*, viene letto dal file specificato l'elenco dei mittenti rifiutati e nel metodo *rate()* viene semplicemente implementato il controllo della presenza del mittente del messaggio all'interno della lista:

```
package it.bigatti.antispam;
import java.io.*;
import java.util.*;
import javax.mail.*;
class FileBlackListRule extends FileBasedRule {
List rejectedSenders;
public FileBlackListRule( String filename )
throws FileNotFoundException, IOException {
rejectedSenders = loadFile( filename );}
public float rate( Message message )
throws MessagingException, RuleException {
if( rejectedSenders.contains( getSender( message ) ) ) {
   return Rule.REJECTED;
} else { return Rule.NONE; } }}
```

La realizzazione di una regola basata su *white list* è invece speculare, con la sola differenza del valore ritornato, che per questa regola varrà *ACCEPTED.*

Per valutare invece la priorità di un messaggio, è necessario interagire più direttamente con le API di *Javamail*, in particolare accedendo alle intestazioni del messaggio per recuperare il valore di quella (o quelle) denominata *X-Priority*. Il metodo *getHeader()* della classe *Message* serve proprio per estrarre, sotto forma di array di stringhe, il contenuto delle intestazioni indicate, se presenti. Il valore ritornato è un array di stringhe perchè una stessa intestazione potrebbe essere presente più volte. Ad esempio, si osservi questo messaggio di prova:

```
From spammer_99@null.it Sat Dec 20 16:57:57 2003
Return-Path: <spammer_99@null.it>
Delivered-To: max@0
Received: (qmail 23653 invoked by uid 511); 20 Dec
                                   2003 12:05:44 -0000
Received: from spammer_99@null.it by mxavas1.aruba.it
                       by uid 503 with qmail-scanner-1.20rc4
(fsecure: 4.50/2111/fprot(2003-10-15)/avp(2003-10-
   16). spamassassin: 2.60. Clear:RC:0:SA:0(1.2/5.0):.
Processed in 0.133694 secs); 20 Dec 2003 12:05:44 -0000
Received: from unknown (HELO vsmtp2.tin.it)
                                      (212.216.176.222)
by mxavas1.aruba.it with SMTP; 20 Dec 2003
                                          12:05:43 -0000
Received: from Computer-di-Massimiliano-Bigatti.local
            (82.48.89.160) by vsmtp2.tin.it (7.0.019)
id 3FE030400010B68D for max@bigatti.it; Sat, 20 Dec
                                   2003 13:05:44 +0100
Message-ID: <15820815.1071922119368.JavaMail.max
            @Computer-di-Massimiliano-Bigatti.local>
Date: Sat, 20 Dec 2003 13:08:38 +0100 (CET)
From: spammer_99@null.it
To: max@bigatti.it
Subject: Messaggio di prova
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
```

Come si nota, l'intestazione *Received* è presente più volte (anche se questa è una chiave particolare).
La stessa cosa si puà verificare per altre intestazioni.
La classe che implementa il controllo della priorità è la seguente:

```
package it.bigatti.antispam;
import javax.mail.*;
/**
* Nel caso il messaggio abbia priorità alta, potrebbe
                                             essere spam
*/
class PriorityRule extends AbstractRule {
final float RATE = 0.5F;
public float rate( Message message ) throws
                                   MessagingException {
String priority[] = message.getHeader("X-Priority");
if( priority != null ) {
```

```
for( int i=0; i<priority.length; i++ ) {
if( priority[ i ].trim().equals( "1" ) ) {
return RATE; } } }
return Rule.NONE; }}
```

La priorità più alta viene contrassegnata con un 1, mentre la media e quella bassa hanno valore 2 e 3. Si noti l'utilizzo della costante RATE per memorizzare il valore da restituire in caso la regola sia verificata.

ANALIZZARE IL CONTENUTO HTML

Si potrebbe pensare che Swing sia una API prettamente orientata al lato client e che in un framework che svolge un servizio, come quello qui descritto, non possa avere niente a che vedere con questa parte della piattaforma Java. Spesso questo è vero ma, quando si ha a che fare con HTML, anche Swing può dare il suo apporto, grazie al parser che integra nel package *javax.swing.text.html*. Nella regola *HasRemovePageRule* è necessario individuare eventuali link a pagine di rimozione da un qualche servizio, identificate dalla presenza della parola "remove" nel link (non è questa l'unica logica possibile, ma solo quella adottata in questo esempio). Per valutare il testo HTML è necessario creare un *DocumentParser* a cui passare il DTD di riferimento (in questo caso "*html*") e su cui chiamare il metodo *parse()*, che si occupa di interpretare il documento. A questo è necessario passare un oggetto *Callback* che ha lo scopo di ricevere notifica per ciascun elemento HTML riscontrato nel documento:

```
DocumentParser parser = new DocumentParser(
                              DTD.getDTD( "html" ) );
RemoveFinderCallback callBack = new
                              RemoveFinderCallback();
parser.parse( new StringReader( text ), callBack, true );
```

La nostra *RemoveFinderCallback* si occupa di individuare i tag *A (<a href="">)* e di verificare che il contenuto dell'attributo href contenga la parola remove. In caso positivo viene impostata la variabile rate, che verrà ritornata tramite il metodo *getRate():*

```
class RemoveFinderCallback extends
                    HTMLEditorKit.ParserCallback {
float rate = Rule.NONE;
public void handleSimpleTag( HTML.Tag tag,
MutableAttributeSet attributes, int pos ) {
if( tag.equals( HTML.Tag.A ) ) {
String href = (String)
attributes.getAttribute( HTML.Attribute.HREF );
if( href != null ) {
if( href.toLowerCase().indexOf( "remove" ) > 0 ) {
rate = RATE; } } } }
```

```
public float getRate() {
return rate; }}
```

IL MOTORE ANTISPAM

Per coordinare l'operatività delle singole regole, è stata realizzata la classe *RuleEngine*, che presenta i metodi pubblici:

- *public static RuleEngine getInstance() throws RuleException*. Questo metodo ritorna una istanza del motore (è un singleton); se l'inizializzazione non avviene con successo viene sollevata una eccezione specifica: *RuleException;*
- *public void setJunkLevel(float level) throws IllegalArgumentException*. Imposta il livello sopra il quale un messaggio viene considerato spazzatura; se il livello impostato è al di fuori del range impostato nella classe viene sollevata una *IllegalArgumentException;*
- *public boolean isJunkMail(Message message) throws RuleException, MessagingException.* Questo metodo valuta un messaggio per indicare se è spazzatura o meno. Può sollevare una eccezione relativa alle regole, oppure alle API Javamail.

Le regole vengono inizializzate dal metodo *initRules()* e mantenute all'interno di una lista:

```
void initRules() throws RuleException {
try {
rules = new LinkedList();
if( isFileWhiteListEnabled() ) {
rules.add( new FileWhiteListRule( "whitelist.txt" ) ); }
if( isFileWhiteListEnabled() ) {
rules.add( new FileBlackListRule( "blacklist.txt" ) ); }
rules.add( new PriorityRule() );
rules.add( new SenderHasUnderlineAndNumbersRule() );
rules.add( new HasRemovePageRule() );
} catch( Exception ex ) {
throw new RuleException(
            "Impossibile inizializzare le regole", ex );}}
```

Il cuore delle operazioni viene però svolto dal metodo protetto *computeRateMessage*, che non fa altro che iterare per tutte le regole presenti nella lista creando una sommatoria delle valutazioni. Ovviamente, nel caso una regola rifiuti od accetti immediatamente il messaggio, la valutazione termina con il risultato ottenuto:

```
protected float computeRateMessage(Message message)
throws RuleException, MessagingException {
float result = 0;
Iterator iter = rules.iterator();
```

UN SISTEMA ESPANDIBILE

La caratteristica principale del sistema è quello di essere espandibile, e cioè di consentire una facile aggiunta o rimozione di regole di valutazione, grazie alla dinamicità di Java ed all'uso delle interfacce. Il sistema può dunque essere considerato un framework, in quanto fornisce semplicemente il servizio di punteggio di un messaggio di posta. Sarà poi un programma specifico che si occuperà di utilizzare questo punteggio.

```
while( iter.hasNext() ) {
Rule rule = (Rule)iter.next();
float rate = rule.rate( message );
if( rate == Rule.ACCEPTED || rate == Rule.REJECTED ) {
result = rate;
break; }
result += rate; }
return result;}
```

UTILIZZARE IL FRAMEWORK

Una volta realizzato il codice che implementa ciascuna regola che si desidera inserire (e magari realizzato anche il codice di test dell'unità per verificarne il corretto funzionamento) è possibile applicare queste regole in pratica. La classe *MailboxScanner* implementa un semplice analizzatore di caselle di posta - in questo caso con protocollo pop3 - presentando a video l'elenco dei messaggi identificati come posta spazzatura ed indicando il relativo punteggio. I tre parametri da passare alla classe sono:

- l'host a cui collegarsi (p.e. *mail.provider.it*);
- il nome utente
- la password.

Il costruttore della classe esegue tutte le operazioni necessarie: per prima cosa si connette al server tramite il metodo *connect()*, poi apre la mailbox principale con *openInbox()* (si noti che con il protocollo *pop3* la casella di posta in arrivo ha il nome standard INBOX). Poi individua con *findJunkMail()* la posta spazzatura, per poi stamparla con *dumpJunkMail()*. La stampa è minimale e prevede la presentazione del mittente, dell'oggetto e del punteggio.

```
package it.bigatti.antispam;
import java.util.*;
import javax.mail.*;
import javax.mail.internet.*;
public class MailboxScanner {
float[] ratings;
public MailboxScanner( String host, String user, String
                                              password )
throws MessagingException, RuleException {
Store store = connect( host, user, password );
Folder inbox = openInbox( store );
if( inbox != null ) {
dumpJunkMail( findJunkMail( inbox ) );}
inbox.close(false);
store.close(); }
Store connect(String host, String user, String password)
throws MessagingException {
Session session = Session.getDefaultInstance(
                             System.getProperties(), null );
Store store = session.getStore("pop3");
store.connect( host, user, password );
return store; }
Folder openInbox( Store store ) throws
                                 MessagingException {
Folder folder = store.getDefaultFolder();
if (folder != null) {
folder = folder.getFolder("INBOX");
if (folder != null) {
folder.open(Folder.READ_ONLY); } }
return folder; }
List findJunkMail( Folder folder )
throws MessagingException, RuleException {
List list = new LinkedList();
Message[] messageList = folder.getMessages();
ratings = new float[ messageList.length ];
RuleEngine ruleEngine = RuleEngine.getInstance();
//usiamo regole molto stringenti...
ruleEngine.setJunkLevel( RuleEngine.MIN_JUNKLEVEL );
int k = 0;
for (int i = 0; i < messageList.length; i++) {
Message message = messageList[ i ];
if( ruleEngine.isJunkMail( message ) ) {
list.add( message );
ratings[ k++ ] = ruleEngine.rateMessage( message );}
}
return list; }
void dumpJunkMail( List list ) throws MessagingException {
System.out.println("Nella tua casella di posta sono
    stati rilevati "+"i seguenti messaggi con punteggio
                           SPAM diverso da zero:");
int i = 0;
Iterator iter = list.iterator();
while( iter.hasNext() ) {
Message message = (Message)iter.next();
System.out.println("Oggetto: " + message.getSubject() +
              " Data: " + message.getSentDate() +
                  " SPAM Rating: " + ratings[ i ]);
i++;}}
public static void main(String args[]) {
try {
new MailboxScanner( args[0], args[1], args[2] );
} catch (Exception ex) {
ex.printStackTrace();
  }
 }
}
```

CONCLUSIONI

Le regole di valutazione antispam qui presentate sono minimali, ma illustrano le tecniche per accedere a quelle parti del messaggio solitamente analizzate in questo tipo di verifiche. Dovrebbe essere dunque abbastanza semplice la realizzazione di ulteriori regole che completino questo framework antispam.

Massimiliano Bigatti

SUL WEB

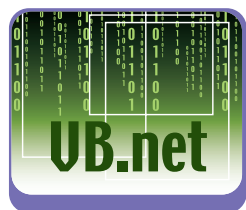
Le mail devono superare un test
www.knowspam.net

I poliziotti del Web
www.spamcop.net

Hai troppi punti? Sei spam!
http://spamassassin.org

Le funzioni per la gestione di un PC in rete

Controllo remoto in VB (parte prima)

In questo articolo realizzeremo un programma Client/Server in grado di effettuare operazioni su macchine remote. Getteremo le basi per realizzare applicazioni distribuite in VB.

Chi ha letto il mio articolo sulla realizzazione di un client di posta in Visual Basic sa benissimo che realizzare un semplice programma in grado di colloquiare con un server qualunque, non è poi così complicato. Il punto importante di questo genere di programmi è senza dubbio la conoscenza degli strumenti messi a disposizione da Visual Basic utili a connettersi a server remoti, oltre alla "padronanza" del protocollo di comunicazione da utilizzare. Ad esempio, se volessimo colloquiare con un server di posta, sarebbe necessario conoscere bene il funzionamento dei protocolli SMTP e POP3. Con un server FTP, invece, dovremmo conoscere il protocollo FTP e così via. In quest'articolo focalizzeremo la nostra attenzione sulla comunicazione tra due programmi, rispettivamente client e server, che si scambieranno informazioni e comandi per ottenere determinati risultati. Ad onor del vero, quindi, tutto quello che dovremo fare è inventarci un protocollo costruito ad hoc ed utilizzarlo alla stessa maniera di quanto già visto nell'articolo menzionato prima.

FUNZIONALITÀ DEL PROGETTO

Il progetto, realizzato in Visual Basic 6, consente di connettersi ad un server realizzato ad hoc ed impartire ad esso comandi di vario genere per ottenere informazioni o effettuare operazioni sul PC remoto. Ovviamente, per quanto appena detto, si evince chiaramente che l'intero programma deve essere suddiviso necessariamente in due parti essenziali: la prima *(client)* in grado di eseguire le operazioni di connessione al computer remoto, ottenere informazioni e chiedere al server di compiere determinate azioni ed una seconda *(server)* in grado di mettersi in ascolto su di una determinata porta TCP/IP (stabilita a priori) per "esaudire" tutte le richieste del client.

Un programma di questo tipo, tanto per citarne uno, è *NetBus* ossia un virus in grado di colloquiare con un modulo server (apparentemente invisibile all'utente) per consentire l'esecuzione di procedure remote, sfogliare i dati presenti sull'hard disk della vittima, ecc. Il progetto presentato in quest'articolo è molto simile a NetBus ma (ovviamente) non è stato realizzato per scopi distruttivi. Attraverso l'analisi del codice presentato, infatti, avremo modo di "ripassare" alcuni importanti aspetti di Visual Basic e vedremo anche come sfruttare alcune tecniche di programmazione molto particolari. Sarà anche un'ottima occasione per vedere alcune API di Windows molto importanti e spesso poco utilizzate.

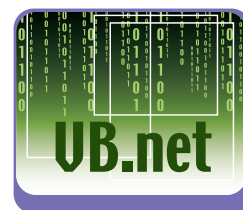
IL CONTROLLO WINSOCK

Attraverso Winsock l'utente/programmatore ha la possibilità di costruire programmi in grado di comunicare con server di qualunque tipo utilizzando protocolli standard (e non) come FTP, HTTP, SMTP, POP3, ecc in maniera piuttosto "semplice". Ovviamente, esistono diversi accorgimenti che vanno adottati quando desideriamo cimentarci in progetti di questo genere e, soprattutto, occorre essere a conoscenza di tutte quelle nozioni, a mio parere fondamentali, che riguardano soprattutto i metodi e le proprietà di questo controllo. A beneficio di chi non lo conosce bene, o crede di saperne poco, analizzeremo gli aspetti fondamentali di questo componente, cercando di soffermarci soprattutto sui punti ritenuti importanti per la corretta comprensione dell'intero progetto. Il controllo Winsock è stato introdotto per consentire una maggiore flessibilità nella costruzione di progetti che prevedono il colloquio tra due PC sfruttando il protocollo TCP/IP o UDP. Questo dettaglio ci obbliga a sottolineare l'esistenza di due possibili modalità di comunicazione:

la prima basata sul protocollo TCP/IP e la seconda basata sull'UDP. Da quanto detto appare evidente che il primo parametro da stabilire quando si decide di mettere in comunicazione due controlli Winsock, è proprio la scelta del protocollo. Una volta che si è deciso il "mezzo" di comunicazione, occorre precisare l'host remoto e le porte alle quali connettersi o rimanere in ascolto. Non dimentichiamoci che un'applicazione client/server prevede da un lato l'avvio in modalità "ascolto" di un socket e, dall'altro, la connessione ad esso attraverso un numero di porta predefinito. Il controllo Winsock permette di creare applicazioni che possono fungere sia da client che da server e tutto questo con un unico "oggetto". Questa doppia funzionalità è posta in atto in maniera semplice attraverso l'impostazione di alcune proprietà. Esso è privo d'interfaccia grafica, ossia è disponibile (visibile) soltanto in fase di design. Cominciamo quindi a spiegare alcune delle proprietà più importanti di questo controllo, peraltro utilizzate all'interno dell'intero progetto:

- **LocalHostName**: restituisce il nome dell'host su cui il componente è in esecuzione;
- **LocalIP:** identifica l'indirizzo IP del computer sul quale è attualmente in esecuzione Winsock;
- **LocalPort:** identifica il numero di porta locale che servirà per la connessione;
- **Protocol:** identifica il tipo di protocollo utilizzato per la connessione. I valori possibili sono rappresentati dalle costanti *sckTCPProtocol* (pari a 0) e *sckUDPProtocol* (pari ad 1);
- **RemoteHost:** identifica il PC con il quale stabilire la connessione. Può assumere valori sottoforma di indirizzo IP oppure sottoforma di nomi host (*FQDN*);
- **RemotePort:** identifica la porta alla quale connettersi;
- **State:** indica lo stato nel quale si trova in un determinato momento il controllo Winsock. Il valore predefinito è rappresentato dalla costante *sckClosed*, pari al valore numerico zero;

Per quanto riguarda i metodi di questo controllo, quelli più importanti sono sicuramente:

- **Accept**: consente di accettare un tentativo di connessione da un computer *(client)* remoto;
- **Close**: chiude la connessione;
- **GetData**: preleva i dati appena ricevuti dal buffer. Questo metodo è molto importante perché consente al server (o al client) di "recuperare" il messaggio del client (o del server), processarlo e comportarsi di conseguenza;
- **Listen**: pone in ascolto un determinato controllo Winsock sulla porta specificata dalla proprietà *LocalPort*. Ovviamente, trattasi in questo caso di applicazioni di tipo *"server"*;
- **SendData**: consente d'inviare una determinata stringa all'host remoto;
- **Connect**: questo metodo innesca il "tentativo" di connessione all'host remoto, attraverso il numero di porta specificato dalla proprietà *RemotePort.*

NOTA

Ecco un elenco dei possibili stati nei quali può trovarsi il controllo Winsock con l'indicazione della costanti che li rappresentano:

COSTANTE	VALORE	DESCRIZIONE
sckClosed	0	Chiuso (default)
sckOpen	1	Aperto
sckListening	2	In ascolto
sckConnectionPending	3	In attesa di connessione
sckResolvingHost	4	Risoluzione dell'host
sckHostResolved	5	Risoluzione dell'host completata
sckConnecting	6	Connessione in corso
sckConnected	7	Connessione completata
sckClosing	8	Connessione in chiusura
sckError	9	Errore

Prima di passare oltre, diamo un brevissimo sguardo agli eventi principali che coinvolgono questo controllo e che interessano il discorso che faremo in seguito:

- **ConnectionRequest:** questo evento è scatenato, lato server, non appena un client tenta di stabilire una connessione con esso;
- **DataArrival:** si verifica quando arrivano nuovi dati;
- **Close:** quest'evento si verifica quando viene chiusa la connessione.

Esistono altre proprietà, metodi ed eventi importanti legati a questo controllo, ma per gli scopi prefissi non verranno menzionati.

BREVE DESCRIZIONE DEL PROGETTO

Inizieremo questo cammino partendo da una breve descrizione dei due moduli realizzati per poi passare ad analizzare nel dettaglio l'intero codice.
Il progetto relativo al server è costituito da quattro form e da due moduli. In particolare:

- **frmAbout:** questa form mostra solo alcune informazioni sul nome del programma, la versione, ecc. Pertanto, può essere modificata senza alcuna conseguenza importante;
- **frmChat:** questa form rappresenta la finestra attraverso la quale sono inviati e ricevuti i messaggi durante una chat;
- **frmServer:** rappresenta la form principale del

progetto;

- **frmTNA:** questa form è stata introdotta per la gestione della *Tray Notification Area* (TNA);
- **Generale.bas:** raccoglie tutte le funzioni e le subroutine utilizzate dal programma;
- **Share.bas:** raccoglie tutte le funzioni e le subroutine (principalmente API di Windows) per effettuare una condivisione delle risorse.

Fig. 1: *L'interfaccia grafica del server.*

All'interno della form principale sono stati sistemati tre controlli Winsock che serviranno a gestire rispettivamente la comunicazione *(chat)* tra i due PC, il download di file (il programma prevede solo l'invio del file contenente la schermata visibile sul PC remoto in un determinato istante) e l'invio di informazioni varie oltre all'esecuzione di determinati comandi. Sempre sulla stessa form sono stati aggiunti tre controlli *Listbox* che consentono, in ogni momento, di avere un'idea di quello che sta accadendo. In queste tre listbox verranno elencate diverse informazioni, tra cui proprio i comandi che il client ha appena impartito al server. All'avvio del server vengono impostati tutti i parametri utili alla comunicazione con il client e viene posto in stato *"listen"* il controllo *WinsockServer*. Successivamente, il modulo è "confinato" all'interno della TNA e resta in attesa di una connessione da parte del client. Il modulo client, invece, è costituito da una sola maschera e da un solo modulo, *Generale.bas* che raccoglie tute le funzioni e le subroutine utilizzate nel programma. Il form principale, *Main*, è costituito da diversi controlli. Innanzitutto, similmente a quanto visto per la parte server, sono stati previsti tre analoghi controlli Winsock. Attraverso essi, a seconda del tipo di operazione che si desidera svolgere, verrà consentita la comunicazione con il server che gestirà opportunamente le varie richieste. L'intera form è costituita da tre sezioni principali. Nella parte superiore troviamo tutto l'occorrente per connetterci (o disconnetterci) ad un PC remoto. Nella parte sinistra, invece, troviamo diversi pulsanti. Ognuno di essi, se premuto, rende visibile un apposito pannello, a destra di questi controlli, che consente d'impartire gli opportuni comandi previsti o analizzare le informazioni che abbiamo richiesto. Come già anticipato, la terza sezione della form è costituita da una serie di pannelli sovrapposti che vengono resi "visibili" a seconda del pulsante premuto. In sintesi, ecco le operazioni che possono essere compiute attraverso i *CommandButton* citati:

- Inviare un messaggio a video, di qualunque tipo, al PC remoto;
- Comunicare attraverso chat;
- Ottenere un file con l'attuale *"Print Screen"* del PC remoto;
- Effettuare diverse operazioni come: aprire o chiudere la porta del lettore CDRom, effettuare il logoff, avviare diversi pannelli di controllo, lanciare una qualunque applicazione, ecc;
- Ottenere alcune informazioni come nome del PC, nome utente, variabili di ambiente, ecc;

Possiamo facilmente intuire come tutto ciò sia possibile. Il modulo client, a seconda dell'operazione che deve effettuare, impartisce l'opportuno comando al server. Quest'ultimo, una volta che lo ha ricevuto, lo processa ed effettua la relativa operazione. Ovviamente, come già spiegato all'inizio, il protocollo di comunicazione è stato costruito ad hoc e non ha nulla a che fare con quelli definiti "standard" come l'SMTP o l'HTTP. Potete modificarlo o ampliarlo a vostro piacimento se lo ritenete opportuno, ricordando però che, se effettuate una modifica di questo genere, essa deve ripercuotersi necessariamente anche sul server o quest'ultimo non riconoscerà affatto "quello che gli dite". A questo punto possiamo passare a descrivere i due progetti, iniziando dal client.

L'AVVIO DEL SERVER

L'avvio del modulo server ha come effetto iniziale quello di scatenare l'evento *Load()* della form *Main*. Al suo interno è inserito il seguente codice:

```
lblHostID.Caption=WinsockServer.LocalHostName
lblAddress.Caption=WinsockServer.LocalIP
WinsockServer.LocalPort=PORT_RC
WinsockServer.Protocol=sckTCPProtocol
WinsockChat.LocalPort=PORT_RC+1
WinsockChat.Protocol=sckTCPProtocol
WinsockFTP.LocalPort=PORT_RC+2
WinsockFTP.Protocol=sckTCPProtocol
LogMsg="In ascolto porta: " & WinsockServer.LocalPort
EventLog.AddItem (LogMsg)
'Mettiti in ascolto (solo con WinsockServer).
'Ad autenticazione avvenuta, effettua la stessa operazione
'con i restanti Winsock
  WinsockServer.Listen
'Confina all'interno della TNA
  Add_Icon
End Sub
```

Le istruzioni precedenti non fanno altro che inizializzare le proprietà principali dei controlli *Winsock* della form, porre in stato *Listen* il controllo *WinsockServer* e confinare il modulo all'interno della *System Tray*. Quest'ultima operazione è compiuta attraverso la chiamata alla procedura *Add_Icon()*, contenuta all'interno del modulo *Generale.bas*, che

si preoccupa di svolgere le necessarie operazioni utili a questo scopo:

```
'Riempiamo la struttura utile alla Tray Notification Area
'La sua dimensione, in byte
  GICO.cbSize=Len(GICO)
'Il puntatore alla finestra che riceve il messaggio di notifica
  GICO.hwnd=frmTNA.PicTNA.hwnd
'Il "nome" dell'icona
  GICO.uID=100
'L'operazione da eseguire:
'   -aggiungere un'icona,
'   -l'ID del messaggio (nome) di notifica,
'   -ed un tooltip
  GICO.uFlags=NIF_ICON Or NIF_MESSAGE Or NIF_TIP
  GICO.uCallbackMessage=WM_MOUSEMOVE
'Picture nella TaskBar
  GICO.hIcon=frmTNA.PicTNA.Picture
'Tooltip di aiuto
  GICO.szTip="Remote Client 1.0 - by Io
                                    Programmo"+Chr$(0)
'Dichiaro alla shell, la presenza dell'icona
  Shell_NotifyIconA NIM_ADD, GICO
  frmServer.Hide
End Sub
```

Poiché non è tra gli scopi di questo articolo spiegare come avviene la gestione della *Tray Notification Area*, tralasceremo di spiegare questo argomento nei dettagli, lasciando che ognuno di voi si preoccupi di comprendere il significato del codice poc'anzi presentato. Solo per chi non ha mai avuto a che fare con questo tipo d'implementazione, possiamo aggiungere che la parte essenziale di questo frammento di codice consiste nel riempire "opportunamente" una struttura di tipo *NOTIFYICONDATA* (nel progetto viene dichiarata una struttura di questo genere denominata *GICO*) richiamando successivamente l'API *Shell_NotifyIconA()* con i due parametri *NIM_ADD* e *GICO*. Inoltre, si presti attenzione all'item *hwnd* di tale struttura che altro non è che l'handle della "finestra" che dovrà gestire i movimenti e le azioni del mouse compiuti sull'icona confinata in *TNA*. All'interno del progetto corrente questo parametro corrisponde all'handle dell'unica picturebox presente all'interno della form *frmTNA*.
Di seguito, per maggiore chiarezza, è mostrata la struttura *NOTIFYICONDATA*:

```
Type NOTIFYICONDATA
    cbSize As Long
    hwnd As Long
    uID As Long
    uFlags As Long
    uCallbackMessage As Long
    hIcon As Long
    szTip As String * 64
 End Type
```

LA CONNESSIONE CON IL SERVER

Una volta che il modulo server è stato avviato sulla macchina remota e siamo certi di essere in grado di comunicare con essa via TCP/IP, possiamo avviare il client e connetterci ad esso. Per far questo è sufficiente inserire l'indirizzo IP di questa macchina all'interno della textbox presente nella parte superiore della form *Main* e premere successivamente il primo dei due pulsanti posti alla sua sinistra. Per ovvie ragioni, all'avvio del programma, tutti i controlli "visibili" sono disabilitati. L'unico abilitato è proprio il pulsante *cmdConnect* che consente la connessione. La pressione di questo pulsante scatena una serie di operazioni importanti ossia:

Fig. 2: *L'interfaccia grafica del client*

- Controllo sull'inserimento di un indirizzo IP nell'opportuna textbox;
- Inizializzazione delle proprietà fondamentali per la comunicazione (relativamente ai tre controlli *Winsock*);
- Invio di un messaggio *RCKNOW*;
- Invio di un messaggio *RCPASS* seguito dalla *"password"* ioProgrammo;
- Connessione, in caso di autenticazione eseguita, di tutti i Winsock ed abilitazione di tutti i controlli visibili della form.

In Visual Basic tutte queste operazioni corrispondono al codice inserito all'interno dell'evento *Click()* di *cmdConnect* e precisamente:

```
'Controlla che sia stato inserito un IP Address.
'In caso contrario, avverti con un messaggio
   If txtConnect.Text="" Then
      MsgBox "Inserisci l'IP Address del PC a cui
                                  collegarti!", vbCritical
      txtConnect.SetFocus
      Exit Sub
   End If
  'Imposta i dati dell'host remoto...
   WinsockClient.RemoteHost=txtConnect.Text
   WinsockClient.Protocol=sckTCPProtocol
   WinsockClient.RemotePort=PORT_RC
   WinsockClient.LocalPort=0
   WinsockChat.RemoteHost=txtConnect.Text
   WinsockChat.Protocol=sckTCPProtocol
   WinsockChat.RemotePort=PORT_RC+1
   WinsockChat.LocalPort=0
   WinsockFTP.RemoteHost=txtConnect.Text
   WinsockFTP.Protocol=sckTCPProtocol
```

```
   WinsockFTP.RemotePort=PORT_RC+2
   WinsockFTP.LocalPort=0
'Collegati...
   If WinsockClient.State <> 0 Then
      WinsockClient.Close
   End If
   WinsockClient.Connect
'Assicurati che la risposta provenga dal server giusto...
'La stringa da controllare deve essere uguale
'alla costante WELCOME_RC definita nel modulo server
   If Not WaitFor(RispostaSRV, "Remote Client 1.0 -
                  by ioProgrammo", RispostaPREC) Then
      MsgBox "Errore di autenticazione su RCKNOW"
      Exit Sub
   End If
'Richiedi autenticazione
   WinsockClient.SendData "RCKNOW"+vbCrLf
   If Not WaitFor(RispostaSRV, "+OK", RispostaPREC) Then
      MsgBox "Errore nel tentativo di connessione a
                                      WinsockClient"
      WinsockClient.Close
      Exit Sub
   End If
'Autenticati con la password
   WinsockClient.SendData "RCPASS "+"ioProgrammo"+vbCrLf
   If Not WaitFor(RispostaSRV, "+OK", RispostaPREC) Then
      MsgBox "Errore di autenticazione su PASS"
      WinsockClient.Close
      Exit Sub
   End If
'Collegati alla chat
   WinsockChat.Connect
   If Not WaitFor(RispostaChat, "Remote Client 1.0
       (CHAT) - by ioProgrammo", RispostaPREC) Then
      MsgBox "Errore nel tentativo di connessione a
                                        WinsockChat"
      Exit Sub
   End If
'Collegati all'ftp
   WinsockFTP.Connect
   If Not WaitFor(RispostaFTP, "Remote Client 1.0
         (FTP) - by ioProgrammo", RispostaPREC) Then
      MsgBox "Errore nel tentativo di connessione a
                                         WinsockFTP"
      Exit Sub
   End If
'Se tutto è andato OK, allora abilita i pulsanti
'per il controllo remoto
   AbilitaPulsanti (True)
End Sub
```

Fig. 3: ***La comunicazione tra il client ed il server.***

Il codice appena illustrato è abbastanza semplice da comprendere. Innanzitutto, la porta di comunicazione del primo socket, è impostata secondo il valore di una costante denominata *PORT_RC*. Questo accorgimento ci consente di variare il valore delle porte a cui collegarsi intervenendo soltanto sulla costante appena menzionata. Così facendo si evita di dover modificare la proprietà *RemotePort* di ciascun controllo Winsock ogni qualvolta desideriamo variare la porta di comunicazione. Per convenzione, il controllo *WinsockClient* si connette alla porta *PORT_RC*, il controllo *WinsockChat* a *PORT_RC+1*, mentre il controllo *WinsockFTP* a *PORT_RC+2*. Successivamente viene tentata la comunicazione con il server attraverso *WinsockClient*. Il primo messaggio inviato è *RCKNOW*. Non appena RCKNOW è ricevuto, il modulo server (ed in particolare, il controllo *WinsockServer*) invia in risposta la stringa *"Remote Client 1.0 - by ioProgrammo"* che garantisce al client di essersi connesso al PC corretto. Se questa fase è andata bene, allora il client invia un secondo messaggio che rappresenta una sorta di autenticazione. Infatti, l'invio della stringa *"RCPASS ioProgrammo"* viene accettata dal server che risponde con un semplice *"+OK"* ed accetta il collegamento. Al momento della ricezione, di questo messaggio di accettazione, vengono attivati finalmente i due restanti controlli *WinsockChat* e *WinsockFTP* con modalità simili a quanto visto precedentemente e vengono abilitati tutti i controlli visibili sulla form attraverso una chiamata alla procedura *AbilitaPulsanti(True)*. Questa routine ha anche un altro compito importante da svolgere, ossia abilitare un controllo di tipo *Timer* che consente di visualizzare graficamente, nella barra di stato posta nella parte inferiore della form, lo stato dei controlli Winsock in ogni momento. Prima di passare oltre, è bene soffermarci qualche istante sulla funzione *WaitFor()*, inserita subito dopo ogni messaggio inviato ad un socket remoto. La funzione *WaitFor()* è assolutamente necessaria perché consente d'impostare i corretti tempi d'attesa tra l'invio di un messaggio al server ed il ricevimento della sua risposta. Infatti, quando il client invia un messaggio qualsiasi al PC remoto, non può di certo aspettarsi una risposta immediata (per ovvi motivi) e, dunque, è necessario introdurre un intervallo di attesa minimo che consenta al server di processare la richiesta e rispondere. La funzione *WaitFor()* è sostanzialmente la stessa vista nell'articolo *"Un client di posta in VB"*, ma qui è stata modificata opportunamente per permetterci di gestire le stringhe di risposta di più Winsock. A questo punto siamo finalmente connessi con il server e possiamo iniziare ad impartirgli i nostri comandi.

CONCLUSIONI

La prossima volta entreremo maggiormente nel merito di questo progetto. Vedremo quali sono le funzionalità reali del client, come si comporta il server al ricevimento di una richiesta e soprattutto avremo l'occasione di rivedere l'utilizzo di alcune API di Windows molto importanti.

Francesco Lippo

Un'applicazione per costruire mosaici con MFC

Realizzare un mosaico di fotografie

parte seconda

Oggetto di questo articolo sarà lo sviluppo di un'applicazione completa per la costruzione di mosaici a partire da un'immagine, assicurandoci che risponda in modo "robusto" alle richieste degli utenti.

Nello scorso numero abbiamo analizzato le problematiche relative alla creazione di un'applicazione professionale. La nostra applicazione, una volta impostati tutti i parametri di calcolo tramite l'interfaccia utente, avvierà un processo di calcolo dell'immagine che potrebbe durare, a seconda del numero di immagini e della loro dimensione, anche diversi minuti. Immaginate se l'utente dovesse aspettare per dieci o più minuti senza sapere né immaginare quando l'applicazione gli mostrerà il risultato di quel calcolo e immaginate quanto fastidioso potrebbe essere il dover annullare quella operazione terminando bruscamente l'applicazione con il celeberrimo ctrl-alt-canc. Per usare il gergo tecnico, vedremo come eseguire i calcoli in maniera concorrente sfruttando le caratteristiche multithreading di Windows per poi addentrarci nel problema squisitamente algoritmico dell'applicazione che stiamo creando. Prima di tutto, vediamo come realizzare un controllo avanzato di visualizzazione immagini con possibilità di zoom rapido e di scrolling, che sarà fondamentale all'interno dell'applicazione e come creare una finestra di dialogo che ci dia la percentuale di avanzamento dell'operazione in corso, il tempo residuo stimato e una descrizione dei vari passaggi del calcolo.

IL CONTROLLO DI VISUALIZZAZIONE IMMAGINE

Per ottenere un buon risultato con la tecnica del fotomosaico dobbiamo lavorare con immagini che abbiano una buona risoluzione e, quindi, dalle dimensioni sicuramente più grandi di quelle del nostro desktop.
La dimensione delle patch (le immagini piccole che devono essere giustapposte) deve essere, infatti, notevolmente più piccola della dimensione dell'immagine sorgente. Per avere un ordine di grandezza si può pensare che dentro all'immagine sorgente ci devono stare almeno quaranta patch in orizzontale e quaranta in verticale. Dovendo lavorare con immagini di una certa dimensione, si presenta la necessità di usare un controllo di visualizzazione immagini che sia in grado di scrollare e zoomare in maniera rapida e intuitiva. Realizzeremo anche questa volta un controllo personalizzato derivandolo direttamente dalla classe MFC *CScrollView* (la quale a sua volta deriva da *CView*) aggiungendovi le funzionalità di scrolling. L'implementazione di questa classe non è certamente cosa da poco, c'è un bel po' di codice da scrivere; guardandone il lato positivo, tutto questo lavoro verrà fatto una sola volta, e ci si troverà il controllo sempre pronto all'uso. *CScrollImagePreview* è la classe che implementa il nostro controllo, al suo interno memorizzeremo i bit delle immagini tramite un oggetto MFC *CImage* del quale sfrutteremo anche le comode funzioni di caricamento dei file JPEG, GIF, BMP e PNG. Per evitare problemi di ritracciamento useremo, nel nostro controllo, il meccanismo del double buffering, imposteremo cioè un contesto di dispositivo di appoggio sul quale apporteremo le modifiche grafiche che, una volta ultimate, verranno copiate in un'unica soluzione a video. Questa tecnica è la stessa usata nei videogame per evitare che su schermo si veda quel fastidioso effetto di "costruzione dell'immagine". Per implementare il double buffering dobbiamo creare, prima di tutto, il contesto di dispositivo nascosto. A tale scopo chiameremo il metodo privato *CreateBackDC* nell'evento *OnCreate* del nostro controllo. Il contesto di dispositivo, che da adesso chiameremo per semplicità dc, deve essere in tutto e per tutto conforme al dc a video in maniera da non alterarne i colori durante il blitting. Nel metodo *CreateBackDC* prendiamo quindi le dimensioni del *dc* originale e, tramite il metodo *CreateCompatibleDC* diamo origine a un *dc*. A quest'ultimo assegneremo, tramite il metodo *SelectObject*, una bitmap delle dimensioni corrette e creata con la funzione *Create-*

Il multithreading è una tecnica di programmazione che consente l'avvio di più "percorsi" di calcolo contemporanei nell'ambito della stessa applicazione.

NOTA

Quando un'applicazione avvia dei processi di calcolo di una certa consistenza è bene che il software dia all'utente un'idea di quanto questi possono durare. Solitamente, a tale scopo, vengono usati i controlli "barra di progressione". I programmi più raffinati mostrano anche il tempo residuo in ore, minuti e secondi.

CompatibleBitmap. Ecco il codice:

```
CRect wndRect;
GetWindowRect(&wndRect);
CDC *pDC = GetDC();
m_BackDC.CreateCompatibleDC(pDC);
m_BackBmp.CreateCompatibleBitmap(pDC,
                        wndRect.Width(), wndRect.Height());
m_pOldBmp = m_BackDC.SelectObject(&m_BackBmp);
m_BackDC.SetStretchBltMode(COLORONCOLOR);
ReleaseDC(pDC);
```

Notiamo come, nell'assegnare la nuova bitmap al *dc*, conserviamo il puntatore alla vecchia bitmap in modo da ripristinarla prima di deallocare il *dc*. In questo modo non avremo nessuna perdita di memoria. Infine chiamiamo il metodo *SetStretchBltMode* settando la modalità di stretching in maniera da evitare che venga eseguito l'antialiasing quando le immagini vengono ridotte di dimensioni.
Chiameremo *CreateBackDC* anche nell'evento *OnSize* in modo da mantenere le dimensioni del buffer coerenti con quelle del controllo.
L'immagine visualizzata nel nostro controllo può essere sia ingrandita che ridotta: in particolare, quando l'immagine diventa più piccola del controllo stesso, dobbiamo visualizzare una cornice nera intorno ad essa. Implementeremo tale cornice nell'override del metodo *OnEraseBackGround* il quale disegna un rettangolo nero grande quanto il controllo. Possiamo utilizzare questo metodo, grazie al fatto che il nostro controllo utilizza il double buffering, in caso contrario avremmo infatti prodotto un fastidioso ritracciamento dell'immagine. Come potrete notare dal codice è stato definito anche il metodo *Create* in modo da lasciare che l'utente specifichi solo i parametri veramente necessari alla creazione del controllo, ovvero: le sue dimensioni, la finestra genitore e un identificativo. Il metodo *Create* di base accetta, infatti, molti più parametri dei quali non necessitiamo. Quando l'immagine verrà visualizzata per la prima volta le sue dimensioni verranno adattate a quelle del controllo in modo da visualizzarla quanto più grande possibile. È importante però fare attenzione all'aspect ratio, non dobbiamo cioè alterare le proporzioni dell'immagine, di questo si occuperà il metodo *ZoomToFitRectangle* nel quale dobbiamo decidere se adattare l'immagine in orizzontale o in verticale e calcolare l'altra dimensione di conseguenza:

```
int distW = (rect.Width() - clientRect.Width());
int distH = (rect.Height()- clientRect.Height());
distW = distW * rect.Height() / rect.Width();
if (distW > distH) SetZoomPercentage(100.0f *
                              clientRect.Width()/rect.Width());
 else SetZoomPercentage(100.0f * clientRect.Height()
                                          / rect.Height());
```

Come possiamo vedere, una volta riadattata l'immagine, viene determinata la percentuale di zoom. Il metodo *ZoomToFitRectangle* verrà usato anche per zoomare un rettangolo dell'immagine dopo averlo selezionato trascinando il mouse con il tasto sinistro premuto. Questa funzione risulta molto utile per evidenziare velocemente i dettagli dell'immagine.
Il nostro controllo permette di zoomare dall'1% al 1600% dell'immagine originale, tale zoom però non sarà lineare, non verrà cioè incrementato in maniera costante ogni volta che si ingrandirà o ridurrà. Ciò dipende dal fatto che, all'aumentare dello zoom, lo scatto successivo potrebbe diventare troppo grande o troppo piccolo.
Precedentemente abbiamo parlato di adattare le dimensioni dell'immagine a quelle del controllo: quando si effettua questo tipo di operazione la percentuale di zoom non necessariamente corrisponderà ad uno dei 24 valori che abbiamo appena specificato. Cambiando il livello di zoom, verrà dunque cercato il valore più simile alla percentuale corrente e, da questo, verrà ottenuto il successivo o il precedente livello di zoom. Questa operazione viene eseguita in entrambi i metodi *SafeIncreaseZoom* e *SafeDecreaseZoom*. Abbiamo così realizzato un potente controllo di visualizzazione immagini. A partire da esso ne deriveremo un secondo: *CProjectImagePreview* il quale, ridefinito il metodo *OnDraw*, sarà in grado anche di disegnare una griglia e di mantenerla coerente con il livello di zoom. Tale griglia rappresenta la suddivisione dell'immagine in corrispondenza della quale verranno applicate le patch.

LA FINESTRA DI PROGRESSO

Come abbiamo più volte ribadito, ogni volta che dobbiamo avviare un'operazione che richiede considerevoli tempi di calcolo, dovremmo permettere all'utente di conoscere quanto tempo occorrerà e, eventualmente, consentire l'annullamento dell'operazione in corso. Implementeremo una finestra di dialogo generica che aiuterà l'utente ad avere un'idea delle operazioni che stiamo eseguendo. Tale finestra presenterà, in particolare, una listbox che mostra una descrizione dell'operazione corrente, una progressbar che presenta lo stato di avanzamento, un'etichetta di testo che mostra il tempo trascorso e il tempo residuo, e un bottone che permette di annullare l'operazione in corso (Fig. 2). Quando vogliamo avviare un thread di calcolo dobbiamo, come prima cosa, dichiarare e implementare la funzione che rappresenta il thread come funzione a se stante. Ciò è necessario dal momento che i metodi di classe non possono essere lanciati come thread.
A causa di questo distacco dal resto del codice, è conveniente implementare un sistema per chiama-

re i metodi della finestra di progresso senza conoscerne il puntatore. In pratica la finestra di progresso verrà istanziata una sola volta e verrà memorizzato il puntatore alla sua istanza in un membro statico della classe stessa. In questo modo basterà chiamare da qualunque funzione il metodo statico *GetInstance* per ottenere il puntatore alla finestra di progresso e utilizzarne i metodi come in questo esempio:

```
CProgressDlg::GetInstance()->Activate("Counting
                                    patch images...");
```

L'istanza vera e propria della finestra di progresso verrà creata all'inizio del programma e gli verrà passato come parametro il puntatore alla finestra principale della nostra applicazione. Grazie a tale puntatore, la finestra di progresso sarà in grado di rendersi modale quando ne verrà chiamato il metodo *Activate* semplicemente disabilitando la finestra principale e mostrandosi in primo piano. Quando l'elaborazione sarà terminata bisognerà chiamare il metodo *Deactivate* che provvederà a nascondere la finestra di progresso e a mostrare nuovamente la finestra principale. Abbiamo anche parlato di tempo residuo il cui calcolo avviene all'interno della funzione *SetProgressPos*. Questo metodo si occupa anche di impostare il progresso corrente e di aggiornare coerentemente la barra di progresso. Il tempo residuo viene stimato con una proporzione e in base al tempo impiegato per arrivare al progresso corrente. È da notare il fatto che questo approccio risulta piuttosto impreciso all'avvio del calcolo e diventa sempre più preciso man mano che il thread di calcolo viene eseguito. Ciò è dovuto al fatto che l'avanzamento del progresso non avviene, solitamente, in maniera costante nel tempo. Il calcolo del tempo residuo si dovrà, quindi, continuamente riadattare alla velocità di avanzamento del progresso. La funzione di calcolo che viene avviata come thread dovrà ricevere come parametro i dati sui quali lavorare. Nel nostro codice, avendo convenientemente accorpato tutti i dati del progetto in un'unica classe progetto, sarà sufficiente passare, come parametro al thread, un puntatore ad un'istanza di tale classe. Una funzione thread, se dichiarata all'interno di una classe, deve essere un metodo *static* e può accettare un solo parametro.
Ecco un esempio di dichiarazione:

```
DWORD WINAPI CProject::RenderImageAreaThread(
                                        LPVOID lpParam)
{ CProject *pPrj = (CProject *)lpParam;
...
```

Come possiamo notare, nella prima riga della funzione otteniamo il puntatore all'istanza della classe progetto che gli era stata passata come parametro.

Ed ecco come avviamo il thread tramite la funzione *CreateThread* delle API di Windows:

```
CreateThread(
  NULL,  // security attributes
  0,     // Default stack size
  MTfunction,  // puntatore alla funzione thread
  this,  // Parametro passato al thread
  0,  // Default creation flags
  &dwThreadId); // Restituisce l'identificatore thread
```

Possiamo notare che il quarto parametro è un puntatore alla classe di progetto, nel nostro caso il suo valore sarà this dal momento che chiameremo *CreateThread* direttamente da un metodo del progetto. Nel software che stiamo realizzando utilizziamo il multithreading solo per mostrare l'avanzamento dei processi di calcolo e per non far risultare bloccata l'applicazione.

Fig. 1: ***La finestra che mostra lo stato di avanzamento dell'operazione in corso.***

Per questo motivo, non è necessario addentrarsi nei sofisticati meccanismi di sincronizzazione degli accessi ai dati che sono tipici dell'approccio multithread. Siamo in possesso di tutti gli strumenti necessari a mettere in piedi l'applicazione, non ci resta che definire nel dettaglio il funzionamento dell'algoritmo di creazione del fotomosaico.

IL FOTOMOSAICO

La creazione di un fotomosaico passa attraverso le seguenti fasi:

1) Scelta dell'immagine da mosaicizzare
2) Scelta delle dimensioni dell'immagine finale e delle patch
3) Scelta dell'archivio di immagini da usare come patch
4) Ridimensionamento delle patch
5) Rendering dell'immagine finale tramite l'applicazione delle patch sull'immagine originale

Nella nostra applicazione sarà possibile scegliere la dimensione delle patch o, alternativamente, scegliere il numero di suddivisioni orizzontali e verticali dell'immagine sorgente (in questo caso la dimensione delle patch verrà calcolata di conseguenza). In questa impostazione saremo aiutati dal controllo di visualizzazione immagine tramite una griglia sovrapposta all'immagine sorgente. Una volta scelta la dimensione delle patch dobbiamo indicare all'applicazione il percorso sul quale trovare le foto da usare. Premendo poi sul tasto "test prepare" esse verranno caricate e ridimensionate in memoria. Fi-

L'antialiasing è una compensazione del deterioramento delle immagini quando queste vengono ridotte. Nel nostro caso, dal momento che le patch vengono ridotte notevolmente, l'antialiasing produce in realtà un effetto peggiore della non compensazione, per questo motivo lo disabiliteremo nelle nostre immagini.

nito questo processo siamo pronti per il rendering, possiamo però scegliere di salvare le patch già ridimensionate in modo da non eseguire questo passaggio una seconda volta. Dal punto di vista implementativo, tutte le immagini verranno memorizzate in oggetti di tipo *CImageEx*, una classe derivata da *CImage* alla quale sono stati aggiunti i metodi di resize, di crop e di calcolo della "distanza" tra due immagini. Le patch verranno memorizzate in un vettore STL di *CImageEx* dopo essere state ridimensionate e convertite a 32 bit per sfruttarne il notevole incremento di prestazioni che offre il metodo *GetPixelEx* rispetto al *GetPixel* di *CImage*. La distanza tra due immagini viene calcolata come somma della distanza dei pixel nelle posizioni corrispondenti; la distanza tra due pixel viene, a sua volta, calcolata usando le componenti del colore (rosso, verde e blue) come se fossero coordinate di punti tridimensionali e ricavando, quindi, la distanza tra i due punti con la seguente formula:

Fig. 2: La copertina di ioProgrammo dopo essere stata trasformata in un fotomosaico.

```
distance = sqrt((float)((red * red) + (green * green) +
                                        (blue * blue)))
```

Nel confrontare le patch con le suddivisioni dell'immagine sorgente bisogna prestare attenzione al fatto che l'immagine sorgente potrebbe non contenere un numero intero di patch in orizzontale e/o verticale. Riguardo alla scelta dell'archivio di immagini usate come patch notiamo che, probabilmente, non tutte le immagini che useremo avranno un rapporto orizzontale/verticale pari a quello della dimensione delle patch. Il ridimensionamento potrebbe quindi portare a una distorsione. Avremmo potuto implementare un meccanismo di cropping, in modo da lasciare inalterate le proporzioni delle patch, ma saremmo incorsi in tagli delle immagini che sarebbero potuti risultare indesiderati. Iniziato il rendering, verranno presi via via tutti i rettangoli rappresentanti la suddivisione dell'immagine sorgente e confrontati con tutte le patch al fine di trovarne la più simile. Durante tale confronto viene sempre conservata la distanza minima trovata fino a quel momento in modo da scartare velocemente le patch che superano tale soglia, anche se il confronto non viene completato. Una volta trovata la patch più simile, il suo ID verrà memorizzato in una matrice implementata dalla classe *CImagePattern* dopo aver appurato che quell'immagine non si ripeta in un certo intorno. Questo accorgimento riduce la scelta di immagini e potrebbe scartare la patch più simile in favore di una meno precisa ma non presente in quel intorno. Tuttavia la ripetizione delle immagini abbassa la qualità generale del risultato. Per capire questo concetto si pensi ad un cielo non troppo nuvoloso nel quale il colore azzurro sarebbe distribuito su una larga area dell'immagine sorgente, se non avessimo evitato la ripetizione delle patch sarebbe stata scelta sempre la stessa patch per tutta l'area occupata dal cielo. Nel pannello di rendering è presente uno slider che permette di scegliere il livello di dettaglio dell'immagine risultante. Muovendo questo controllo cambierà il numero di pixel tra un campionamento e l'altro all'interno delle patch, influenzando così la precisione del calcolo della distanza. Il livello del dettaglio è impostato per default a *due*, in questo modo verranno campionati solo i pixel che stanno sulle colonne e righe pari, ciò significa che saranno eseguiti un quarto dei *GetPixel* sulle patch e un quarto sull'immagine sorgente; nel caso in cui si stiano usando numerose patch la velocità del calcolo aumenterà sensibilmente. Questa caratteristica può essere usata per ottenere un'anteprima veloce del risultato finale. Finito il calcolo verrà scandita la matrice degli ID delle patch ed esse verranno blittate sull'immagine originale per mostrare il risultato finale. Come si è visto il numero di chiamate a *GetPixel* cresce molto rapidamente all'aumentare del numero di immagini usate come patch. Il numero di tali immagini deve però essere elevato per assicurare una buona varietà e un buon risultato. Per questo motivo è stato scelto di ottimizzare la funzione *GetPixel* anche a discapito della notevole quantità di memoria richiesta dalle immagini a 32 bit.

NOTA

Quando si lavora con le funzioni dell'interfaccia grafica di Windows è piuttosto facile dimenticarsi di deallocare bitmap e dc o di rilasciare i contesti di dispositivo acquisiti. Fortunatamente quando MFC lavora in debug, il suo memory manager è in grado di avvertirci qualora la nostra applicazione presenti dei memory leak (perdita di memoria). Tuttavia non sempre è immediato trovare il punto in cui tali perdite di memoria avvengono.

CONCLUSIONE

Concludiamo così il nostro viaggio nella creazione di un'applicazione MFC completa, la quale ha la pretesa di essere progettata e scritta con particolare riguardo alla tecnica e alla funzionalità. Un' occhiata veloce al codice potrebbe non essere sufficiente a capire alcuni meccanismi e alcune scelte che sono state prese nello sviluppo di questa applicazione. Tuttavia uno sguardo più attento vi farà apprezzare il funzionamento delle classi che abbiamo sviluppato e magari vi darà qualche spunto per un approccio più professionale allo sviluppo delle vostre applicazioni. Non mi resta che salutarvi e augurarvi buona programmazione.

Amedeo Margarese

Completiamo il nostro client di posta.

Un Outlook in Java

(parte quarta)

Siamo giunti all'ultima parte di questo tutorial dedicato allo studio delle API JavaMail e alla loro applicazione per lo sviluppo di un semplice client di posta elettronica. Completiamo il nostro progetto!

Questo mese metteremo insieme le singole esperienze svolte nel corso dei tre appuntamenti precedenti, per tirare le somme e per ottenere finalmente un client di posta completo.

SwingMail è il nome dell'applicazione riassuntiva che allestiremo questo mese, sommando i risultati ottenuti di volta in volta nell'arco delle parti precedenti. Lo scopo è realizzare un piccolo client di natura didattica che supporti simultaneamente le opzioni più basilari di un qualsiasi client di posta elettronica. *SwingMail* non archivierà su disco la posta scaricata, ma funzionerà unicamente come lettore dei contenuti conservati sul server. Se da un lato la scelta può apparire una semplificazione del lavoro da svolgere, dall'altro potrebbe essere una delle caratteristiche vincenti di un software realizzato in questa direzione. Grazie alla portabilità di Java, infatti, può essere comodo avere sempre a portata di mano, in pochi KB, un programmino di lettura della posta elettronica che si sostituisca alla più lenta e laboriosa interfaccia via Web che molti servizi offrono.

LA CLASSE SWINGMAIL

SwingMail costituisce la classe principale della nostra applicazione:

```
// SwingMail.java
// Import dalla libreria di Java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;
import java.util.Properties;
// Import dalle API JavaMail.
import javax.mail.*;
// Classe principale dell'applicazione.
public class
  SwingMail
  ...
  // Punto di ingresso del programma.
  public static void main(String[] args) {
    SwingMail mainFrame = new SwingMail();
    mainFrame.show();}}
```

Simultaneamente, questa classe compie più operazioni. Anzitutto contiene il punto di avvio dell'intera applicazione, cioè il metodo *main()*. Il costruttore della classe svolge tre importantissimi lavori:

1. Assembla l'interfaccia del programma, disponendo una lista e dei pulsanti che saranno utili all'utente per interagire con la posta ricevuta e con quella da inviare.
2. Mostra il gestore dei profili realizzato nell'arco della precedente parte di questo tutorial. Dunque la prima cosa che dovrà fare l'utente ad ogni avvio del programma è selezionare il profilo da impiegare. Solo dopo la selezione di un profilo valido sarà possibile impiegare *SwingMail*.
3. Basandosi sulle informazioni tratte dal profilo selezionato, il costruttore stabilisce una sessione *JavaMail* che sarà conservata all'interno di una proprietà statica della classe. Questa sessione durerà tanto quanto il programma stesso, e le altre classi che a breve analizzeremo potranno accedervi liberamente per fare uso delle caratteristiche di *JavaMail*.

Per il resto *SwingMail* ricicla gran parte del codice osservato nella seconda parte di questo tutorial, quando venne sviluppata la semplice applicazione

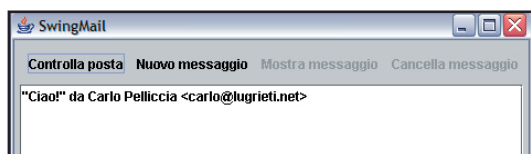



Fig. 1: *L'applicazione SwingMail così come si presenta dopo aver riscontrato la presenza di un nuovo messaggio nella casella di posta elettronica remota.*

MailReceiver, per la visualizzazione della posta ricevuta su un host POP3. La classe interna *ReceiveDialog,* infatti, cura la connessione all'host remoto ed il download delle intestazioni delle mail ricevute. L'elenco della posta in arrivo sarà mostrato nella *JList* che occupa il centro della finestra. L'unica reale novità rispetto al software esaminato qualche mese fa è costituita dall'inserimento di un'opzione utile per cancellare i messaggi conservati dal server. Scorrendo il codice, all'interno del metodo *actionPerformed(),* è possibile localizzare la sezione che si occupa effettivamente di questa operazione:

```
// Cancella il messaggio selezionato.
try {
  int sel = inbox.getSelectedIndex();
  messages[sel].setFlag(Flags.Flag.DELETED, true);
  // Riformula la lista dei messaggi.
  Message[] aux = new Message[messages.length - 1];
  int c = 0;
  for (int i = 0; i < messages.length; i++) {
    if (i != sel) {aux[c++] = messages[i];} }
  // Preparo le intestazioni da mostrare nella lista.
  messages = aux;
  String[] lines = new String[messages.length];
  for (int i = 0; i < messages.length; i++) {
    lines[i] = "\"" + messages[i].getSubject() +
    "\" da " + messages[i].getFrom()[0];}
  // Mostro i messaggi nella lista.
  inbox.setListData(lines);
} catch (Exception e1) {}
```

L'eliminazione avviene alla riga:

```
messages[sel].setFlag(Flags.Flag.DELETED, true);
```

In realtà questo comando non elimina istantaneamente il messaggio dal server, ma lo marca come “messaggio da rimuovere”. La reale eliminazione avviene alla chiusura dell'oggetto Folder collegato. *SwingMail* chiude questo oggetto in due casi: quando si richiede l'aggiornamento della lista della posta in arrivo (lo chiude e lo riapre per rileggerne il contenuto), e quando il programma viene terminato (all'interno del metodo *dispose()*). Ciononostante il messaggio selezionato viene immediatamente rimosso dalla lista dei visualizzati, in modo che l'operazione rimanga trasparente nei confronti dell'utente.

LA CLASSE MESSAGEVIEWER

La classe *MessageViewer* viene richiamata da *SwingMail* ogni volta che l'utente desidera leggere per esteso uno fra i messaggi ricevuti. Anche il codice di *MessageViewer* deriva in gran parte dall'esperienza dell'applicazione *MailReceiver* sviluppata nella seconda parte del tutorial:

```
// MessageViewer.java
// Import dalla libreria di Java.
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import javax.swing.*;
// Import dalle API JavaMail.
import javax.mail.*;
// La classe fornisce un'interfaccia per la lettura dei messaggi.
class MessageViewer extends JFrame implements
                                        ActionListener {
  // Pulsanti.
  JButton button1 = new JButton("Rispondi");
  JLabel info = new JLabel("");
  // Riferimento al profilo in uso.
  Profile profile;
  // Campi per la risposta.
  String replyTo = "";
  String subject = "";
  // Info per salvare gli allegati automaticamente.
  File tmpDir = null;
  int messageNumber;
  // Costruttore.
  public MessageViewer(Profile p, Message m) throws
                                        Exception {
    // Richiamo il costruttore della classe base.
    super("Messaggio");
    // Memorizzo il profilo.
    ...
   // Richiamato alla pressione del tasto "Rispondi".
  public void actionPerformed(ActionEvent e) {
    MailSender sender = new MailSender(profile, replyTo,
                                  "Re: " + subject);
    sender.show();}
  // Salva gli allegati.
  private void save(String name, InputStream in) {...}
}
```

Due sono le principali modifiche apportate: l'inserimento del tasto “Rispondi”, per passare automaticamente alla fase di composizione di un nuovo messaggio, ed una appena differente gestione degli allegati. Ogni volta che una lettera contiene uno o più allegati il software crea una nuova cartella su disco, corrispondente al messaggio stesso, salvando al suo interno i file ricevuti. L'utente viene informato dell'operazione attraverso un avviso eventualmente posto nella parte inferiore della finestra.

PREREQUISITI RICHIESTI

La realizzazione di quanto è mostrato in questo tutorial richiede, da parte del lettore, alcuni prerequisiti di base:

• discreta conoscenza del linguaggio Java e della sua piattaforma;
• discreta conoscenza dei concetti di base del networking;
• discreta conoscenza dei meccanismi tipici di un servizio di posta elettronica.

Nel caso abbiate dei problemi con il terzo requisito richiesto, fareste bene ad approcciare l'argomento prima di intraprendere la lettura dell'articolo. Alcuni link consigliati sono:

http://www.csita.unige.it/servizi/posta/laposta.html
http://www.wowarea.com/italiano/aiuto/abmailit.htm
http://lagash.dft.unipa.it/AL/al263.htm

UN BUON TUTORIAL PER COMINCIARE CON JAVAMAIL

Potete trovare un buon tutorial di base sull'utilizzo delle API JavaMail all'indirizzo:

http://developer.java.sun.com/developer/onlineTraining/JavaMail/

JAVAMAIL FAQ

Un bel po' di risposte alle domande più frequenti su JavaMail le trovate su jGuru, alla pagina:

http://www.jguru.com/faq/JavaMail

Per contattare l'autore di questo articolo scrivi a carlo.pelliccia@ioprogrammo.it

LA CLASSE MAILSENDER

Se gran parte del codice di *SwingMail* e di *MessageViewer* deriva dall'esperienza fatta nella seconda parte di questo tutorial, per lo sviluppo della classe *MailSender* si è attinto a piene mani da quanto studiato e sviluppato nel primo episodio della serie. *MailSender* è un componente utile per la composizione e l'invio della posta attraverso un server SMTP:

Fig. 2: Il componente MessageViewer in azione.

```
// MailSender.java
// Import dalla libreria di Java.
import java.awt.*;
import java.awt.event.*;
import java.io.File;
import java.util.*;
import javax.swing.*;
// Import dalle API JavaMail.
import javax.activation.*;
import javax.mail.*;
import javax.mail.internet.*;
public class MailSender extends JFrame {
  ...
    // Il metodo richiamato per invocare l'invio della mail.
    public void sendMessage() {
      // Avvio il thread che cura l'invio della mail.
      thread = new Thread(this);
      thread.start();
      // Mostro la finestra di attesa.
      show();}
    // L'effettivo invio della mail.
    public void run() {
      // Recupero la sessione.
      Session session = SwingMail.session;
      // Preparo il messaggio da inviare (
                      javax.mail.internet.MimeMessage)
      ...
      // Associo il contenuto al messaggio.
      message.setContent(multipart); }
    // Invio la mail.
    Transport.send(message);
    // Chiudo la finestra d'attesa.
    dispose();
    // Avviso l'utente del successo dell'operazione.
    JOptionPane.showMessageDialog( myself,
```

```
                "La lettera è stata spedita", "Invio eseguito!",
                    JOptionPane.INFORMATION_MESSAGE);
  // Chiudo la finestra del messaggio.
  myself.dispose();
  } catch (Exception e) {
  // Chiudo la finestra d'attesa.
  dispose();
  // Avviso l'utente in caso di errore.
  JOptionPane.showMessageDialog(myself,
                "Impossibile inviare la lettera.\n" +
          "Controllare i campi e ritentare.","Errore!",
                        JOptionPane.ERROR_MESSAGE);
  System.out.println(e);}
  } }
}
```

Le novità di questa versione di *MailSender* sono:

1. Non è più necessario impostare ogni volta il server SMTP e l'indirizzo di posta del mittente, giacché queste informazioni vengono ora tratte dal profilo dell'utente.
2. Il costruttore accetta tra i suoi argomenti due stringe utili per preimpostare il destinatario e l'oggetto della mail. Questa funzionalità è sfruttata quando la classe viene richiamata dall'interno di *MessageViewer*, in modo che alla pressione del tasto "Rispondi" il destinatario e l'oggetto vengano passati da un componente all'altro.

Fig. 3: La finestra di composizione di un nuovo messaggio, realizzata dal componente MailSender.

CONCLUSIONI

Mettendo insieme le tre classi oggi esaminate con il gestore dei profili discusso nell'appuntamento precedente (lo trovate tra i codici della rivista) è possibile compilare il programma e far partire *SwingMail*. Il programma è certamente acerbo, ma costituisce ugualmente un ottimo punto di partenza. Potrete raffinarlo sia per sviluppare un leggero ma più completo gestore che non archivia su disco la posta letta sia per mettere in piedi un client più classico nella direzione di Outlook e Eudora, ma con in più il pregio di essere multipiattaforma.

Carlo Pelliccia

I trucchi del mestiere

Tips & Tricks

La rubrica raccoglie trucchi e piccoli pezzi di codice che solitamente non trovano posto nei manuali, ma sono frutto dell'esperienza di chi programma. Alcuni trucchi sono proposti dalla Redazione, altri provengono da una ricerca su internet, altri ancora ci giungono dai lettori. Chi vuole contribuire potrà inviarci i suoi tips&tricks preferiti che, una volta scelti, verranno pubblicati nella rubrica. Il codice completo dei tips è presente nel CD allegato nella directory *tips*\\ o sul Web all'indirizzo: cdrom.ioprogrammo.it.

COME ESTRARRE LE ICONE DA FILE .EXE O DA LIBRERIE DI ICONE .ICL

Il codice utilizza delle API di sistema, in particolare ExtractIcon e DrawIcon per estrarre e visualizzare l'icona in una finestra di about standard di Windows

Tip fornito dal sig. S.Tomaselli

```
Private Declare Function ExtractIcon Lib "shell32.dll" Alias "ExtractIconA"
(ByVal hInst As Long, ByVal lpszExeFileName As String,
                                        ByVal nIconIndex As Long) As Long
Private Declare Function ShellAbout Lib "shell32.dll" Alias"ShellAboutA"
(ByVal hwnd As Long, ByVal szApp As String, ByVal szOtherStuff As
                                      String, ByVal hIcon As Long) As Long
Private Declare Function DrawIcon Lib "user32" (ByVal hdc As Long,
    ByVal x As Long, ByVal Y As Long, ByVal hIcon As Long) As Long
Private Function Estrai(ByVal NomeFile As String, Index As Long) As Long
                                   Estrai = ExtractIcon(0, NomeFile, Index)
End Function
Private Sub Form_Click()
'Cambiare questa costante se il percorso è diverso.
'In ogni caso indica solamente il file da cui estrarre le icone
'che può essere qualsiasi .exe per windows e qualsiasi file .icl
Const explorer = "c:\windows\explorer.exe"
Dim Icona As Long 'Variabile in cui viene inserito il numero handle
dell'icona estratta
'L'indice dell'icona può essere cambiato, i valori partono da 0
Icona = Estrai(explorer, 5)
DrawIcon Me.hdc, 0, 0, Icona 'Disegna l'icona sul form
ShellAbout Me.hwnd, "NomeProgramma", "", Icona 'Apre la finestra
                                                          di about
End Sub
```

GENERAZIONE DI UNA SEQUENZA CASUALE DI N NUMERI INTERI TUTTI DISTINTI

La funzione proposta genera *n* numeri interi casuali tutti distinti, compresi tra *min* e *max*. Si vuole evitare di generarli uno per volta, per evitare di dover fare, per ciascuno di essi, un ciclo di tentativi.

Tip. Fornito dal sig. R.Bandiera

```
Private Function SequenzaCasuale(ByVal n As Integer, ByVal min As
                  Integer, ByVal max As Integer) As Integer()
` n é il numero di valori desiderati
` min e max sono gli estremi del campo di variazione
` dei valori desiderati
Dim numero() As Integer
Dim i As Integer
Dim j As Integer
Dim appo As Integer
` generazione dei numeri interi da min a max
ReDim numero(1 To max - min + 1)
For i = 1 To max - min + 1
  numero(i) = i + min - 1
Next i
Randomize
` ciclo di mescolamento
For i = 1 To max - min + 1
 j = Int(Rnd() * (max - min + 1)) + 1
 ` scambio tra numero(i) e numero(j)
 appo = numero(i)
 numero(i) = numero(j)
 numero(j) = appo
Next i
` una seconda rimescolata
For i = 1 To max - min + 1
 j = Int(Rnd() * (max - min + 1)) + 1
 ` scambio tra numero(i) e numero(j)
 appo = numero(i)
 numero(i) = numero(j)
 numero(j) = appo
Next i
` restituzione delle prime n componenti
ReDim Preserve numero(1 To n)
SequenzaCasuale = numero()
End Function
```

RICHIAMARE QUERY DA DB ACCESS TRAMITE SELEZIONE DA UNA COMBOBOX

Il codice permette di richiamare, da Visual Basic, delle query contenute in un database Access e di visualizzarle in una tabella in base alla selezione effettuata dall'utente mediante combobox. L'utente selezionerà la voce di suo interesse dalla combobox e premendo il pulsante *cerca* sarà richiamata la query contenuta nel database Access *"Listino"*.

Questo database contiene diverse query, ognuna delle quali è collegata alle voci della combobox. Il codice è presente in formato sorgente nel CD-Rom allegato alla rivista e/o sul sito web di ioProgrammo (*www.ioprogrammo.it*)

Tip fornito dal sig. S.Giolo

```
Private Sub Command1_Click()
Opzione
End Sub
Private Sub Opzione()
If Combo1.Text = "Monitors" Then
Ado.RecordSource = "Monitor"
Set Flex.DataSource = Ado
Ado.Refresh
Else
If Combo1.Text = "Stampanti" Then
Ado.RecordSource = "Stampanti"
Set Flex.DataSource = Ado
Ado.Refresh
Else
If Combo1.Text = "Scanner" Then
Ado.RecordSource = "Scanner"
Set Flex.DataSource = Ado
Ado.Refresh
Else
End If
End If
End If
End Sub
Private Sub Form_Load()
Ado.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data
                                                    Source=" & App.Path &
"\listino.mdb"
Combo1.AddItem "Stampanti"
Combo1.AddItem "Scanner"
Combo1.AddItem "Monitors"
Combo1.ListIndex = 0
End Sub
```

UNA FINESTRA "MAGNETICA"

Il tip produce un effetto "magnetico" sulle finestre durante le operazioni di spostamento delle stesse; un po' come avviene similarmente nell'ambiente KDE di Linux. Per ammirare l'effetto magnetico bisogna attiva l'opzione *Mostra contenuto della finestra* durante l'operazione di trascinamento (dal pannello delle proprietà dello schermo, nella scheda *Effetti*). Il codice è presente in formato sorgente nel cd-rom allegato alla rivista e/o sul sito web di ioProgrammo (*www.ioprogrammo.it*).

Tip fornito dal sig. L. La Marca

```
Public Const GWL_WNDPROC = (-4)
Public Const WM_MOVE = &H3
Public Const ABE_BOTTOM = 3
Public Const ABE_LEFT = 0
Public Const ABE_RIGHT = 2
Public Const ABE_TOP = 1
Public Const ABM_GETTASKBARPOS = &H5
Public Const SWP_NOSIZE = &H1
Public Const SWP_NOZORDER = &H4
Public Const SWP_NOSENDCHANGING = &H400
Public Type RECT
    Left As Long
    Top As Long
    Right As Long
    Bottom As Long
End Type
Public Type APPBARDATA
    cbSize As Long
    hwnd As Long
    uCallbackMessage As Long
    uEdge As Long
    rc As RECT
    lParam As Long
End Type
Public Declare Function SHAppBarMessage Lib "shell32.dll" (ByVal
                    dwMessage As Long, pData As APPBARDATA) As Long
Public Declare Function GetDesktopWindow Lib "user32" () As Long
Public Declare Function GetWindowRect Lib "user32" (ByVal hwnd As
                                        Long, lpRect As RECT) As Long
Public Declare Function GetLastError Lib "kernel32" () As Long
Public Declare Function FindWindow Lib "user32" Alias "FindWindowA"
  (ByVal lpClassName As String, ByVal lpWindowName As String) As Long
Public Declare Function CallWindowProc Lib "user32" Alias
       "CallWindowProcA" (ByVal lpPrevWndFunc As Long, ByVal hwnd
                  As Long, ByVal Msg As Long, ByVal wParam As Long,
                                        ByVal lParam As Long) As Long
Public Declare Function SetWindowLong Lib "user32" Alias
             "SetWindowLongA" (ByVal hwnd As Long, ByVal nIndex
                         As Long, ByVal dwNewLong As Long) As Long
Public Declare Function SetWindowPos Lib "user32" (ByVal hwnd As
       Long, ByVal hWndInsertAfter As Long, ByVal x As Long, ByVal y
          As Long, ByVal cx As Long, ByVal cy As Long, ByVal wFlags
                                                   As Long) As Long
Public trayBar As APPBARDATA
Public lProcOld As Long
Public appRect As RECT, deskRect As RECT
Public Const margine = 20   ' n. di pixel di margine
' Subclassing per la gestione dell'evento WM_MOVE generato
                                    durante gli spostamenti del form
Public Function WindowProc(ByVal hwnd As Long, ByVal uMsg As
        Long, ByVal wParam As Long, ByVal lParam As Long) As Long
    Dim w As Long, h As Long
    Dim move As Boolean
     If uMsg = WM_MOVE Then
       If (GetWindowRect(hwnd, appRect)) Then
           w = appRect.Right - appRect.Left
           h = appRect.Bottom - appRect.Top
           If Not (w < deskRect.Right) And Not (h < deskRect.Bottom) Then
               Exit Function
           End If
           move = False
           If (Abs(appRect.Top - deskRect.Top) <= margine) Then
               appRect.Top = deskRect.Top
               move = True
           End If
           If (Abs(appRect.Left - deskRect.Left) <= margine) Then
```

```
                appRect.Left = deskRect.Left
                move = True
            End If
            If (Abs(appRect.Bottom - deskRect.Bottom) <= margine) Then
                appRect.Top = deskRect.Bottom - h
                move = True
            End If
            If (Abs(appRect.Right - deskRect.Right) <= margine) Then
                appRect.Left = deskRect.Right - w
                move = True
            End If
            If move Then
                SetWindowPos hwnd, 0, appRect.Left, appRect.Top, 0, 0,
                                              SWP_NOSIZE Or SWP_NOZORDER
                                                   'Or SWP_NOSENDCHANGING
            End If
        Else
            Debug.Print GetLastError()
        End If
     End If
     WindowProc = CallWindowProc(lProcOld, hwnd, uMsg, wParam,
                                                               lParam)
 End Function
Public Function GetDesktopRect(ByRef lpRect As RECT) As Long
    Dim sz As RECT
    Dim ret As Long
    trayBar.cbSize = Len(trayBar)
    trayBar.hwnd = FindWindow("Shell_TrayWnd", 0)
    ' calcola le dim. e la posizione della barra degli strumenti
    ret = SHAppBarMessage(ABM_GETTASKBARPOS, trayBar)
    sz = trayBar.rc
    ' calcola le dim. dello schermo
    GetWindowRect GetDesktopWindow(), lpRect
    ' calcola le dim. effettive del desktop
    Select Case trayBar.uEdge
    Case ABE_TOP
        lpRect.Top = sz.Bottom
    Case ABE_LEFT
        lpRect.Left = sz.Right
    Case ABE_RIGHT
        lpRect.Right = sz.Left
    Case ABE_BOTTOM
        lpRect.Bottom = sz.Top
    End Select
End Function
Uso :
...
Private Sub Form_Load()
    GetDesktopRect deskRect
    lProcOld = SetWindowLong(Me.hwnd, GWL_WNDPROC, AddressOf
                                                     WindowProc)
End Sub
Private Sub Form_Unload(Cancel As Integer)
    SetWindowLong Me.hwnd, GWL_WNDPROC, lProcOld
End Sub
...
```

IL TIP DEL MESE

REGISTRARE LE OPERAZIONI COMPIUTE COL MOUSE

Il tip utilizza un *ocx* per registrare i movimenti ed i tasti premuti del mouse; una tale applicazione potrebbe risultare utile in tutte quelle situazioni in cui si rende necessario mostrare, automaticamente, le funzionalità di un programma. Il codice, data la sua prolissità, è presente nel CD-Rom allegato alla rivista e/o sul sito web di ioProgrammo (*www.ioprogrammo.it*)

Tip fornito dal sig. P.Miola

COMUNICAZIONE SERIALE E PARALLELA

Un trucco per aprire, e mettersi in ascolto, su una porta seriale o parallela in Java. Nell'esempio viene mostrato un esempio che utilizza la porta seriale COM1. Per questa operazione è necessario: scaricare il package Java Communication API (*javax.com*) *http://java.sun.com/products /javacomm/index.html* e copiare: *win32com.dll* in *<JDK>\bin*, *comm.jar* in *<JDK>\lib*, *javax.comm.properties* in *<JDK>\lib* e in *<JDK>\jre\lib*. Infine aggiungere *comm.jar* nel classpath:

```
set CLASSPATH=c:\jdk1.1.6\lib\comm.jar;%classpath%
```

Il codice è presente in formato sorgente nel CD-Rom allegato alla rivista e/o sul sito web di ioProgrammo (*www.ioprogrammo.it*)

Tip fornito dal sig. P. Orsini

```
import java.io.*;
import javax.comm.*;
import java.util.*;
public class Com implements Runnable,SerialPortEventListener{
    //Dichiarazioni variabili
    static CommPortIdentifier portId;
     static Enumeration portList;
     SerialPort serialPort;
     Thread readThread;
     private InputStream serialInput;
    public static void main(String[] arg) throws Exception{
        //Richiede la lista delle porte disponibili
        portList=CommPortIdentifier.getPortIdentifiers();
        // Cerca fra le porte disponibili quella richiesta
        //finchè c'è ancora una porta nella lista
        while (portList.hasMoreElements()){
            //Prende un porta dalla lista
            portId=(CommPortIdentifier) portList.nextElement();
            // Controlla che la porta presa in considerazione sia
```

```
        //una porta seriale ed in particolare la COM1
        // Per le porte parallele utilizzare:
        //CommPortIdentifier.PORT_PARALLEL
        if ((portId.getPortType() == CommPortIdentifier.PORT_SERIAL)
            && (portId.getName().equals("COM1"))){
            System.out.println("Porta trovata");
           new Com();
           break;}
     } }
   public Com() throws Exception{
    //Apre la porta con un timeout di 2sec (2000 msec)
    serialPort=(SerialPort) portId.open("RS232",2000);
    //Associa un evento all'input dalla porta
    serialPort.addEventListener(this);
    serialPort.notifyOnDataAvailable(true);
    //Setta i parametri della porta
    serialPort.setSerialPortParams(9600,SerialPort.DATABITS_8,
                    SerialPort.STOPBITS_1,SerialPort.PARITY_NONE);
    serialPort.setFlowControlMode(
                                SerialPort.FLOWCONTROL_RTSCTS_IN);
    //Apre un canale di comunicazione
    serialInput=serialPort.getInputStream();
    System.out.println("Porta aperta");
    //Avvia il thread per l'ascolto
    readThread = new Thread(this);
     readThread.start();}
   //Thread in ascolto sulla tastiera
   public void run(){
    try{
        BufferedReader tastiera=new BufferedReader(new
                                   InputStreamReader(System.in));
        //Attesa comando quit
        while (!(tastiera.readLine().equals("quit")));
        //Chiusura porta
        serialPort.close(); }
    catch (Exception e){
           System.out.println(e);} }
   public void serialEvent(SerialPortEvent event){
    //Verifica la presenza di dati nel buffer
    if (event.getEventType()==SerialPortEvent.DATA_AVAILABLE){
        try{
           //Legge i dati dal buffer (se presenti)
           while(serialInput.available()>0)
                System.out.print((char)serialInput.read()); }
        catch (Exception e){
           System.out.println(e); }
  } }
}
```

DA COMPONENTE A JPEG

Attraverso poche righe di codice è possibile salvare un nostro componente come immagine JPEG. Il codice è presente in formato sorgente nel CD-Rom allegato alla rivista e/o sul sito web di ioProgrammo (*www.ioprogrammo.it*).

Tip fornito dal sig. M.Rapa

```
import java.io.OutputStream;
import java.io.FileOutputStream;
import com.sun.image.codec.jpeg.JPEGImageEncoder;
import com.sun.image.codec.jpeg.JPEGCodec;
import java.awt.Component;
import java.awt.image.BufferedImage;
import java.awt.Dimension;
import java.awt.Graphics2D;
public class ComponentToJPEG {
 /**
  *
  * @param myComponent
  * @param filename
  */
 public static void saveComponentAsJPEG(Component
                                  myComponent, String filename) {
   Dimension size = myComponent.getSize();
   BufferedImage myImage =new BufferedImage(size.width,
                         size.height,BufferedImage.TYPE_INT_RGB);
   Graphics2D g2 = myImage.createGraphics();
   myComponent.paint(g2);
   try {
     OutputStream out = new FileOutputStream(filename);
     JPEGImageEncoder encoder = JPEGCodec.createJPEGEncoder(out);
     encoder.encode(myImage);
     out.close();}
   catch (Exception e) { System.out.println(e);}
}}
```

NASCONDERE IL TESTO RENDENDOLO ILLEGGIBILE

Si tratta di una piccola implementazione di un algoritmo per nascondere del testo in chiaro e renderlo illeggibile, almeno per un po' di tempo. Si basa sullo scambio di posizione e di carattere in modo variabile. La fase di crittazione richiede di inserire il nome del file di testo da trasformare, che dovrà essere presente nella cartella dell'eseguibile, e lo trasforma in "*crypted.dat*". La fase di decrittazione, invece, carica automaticamente tale file ("*crypted.dat*"), che sarà presente nella cartella dell'eseguibile, e lo trasforma in un file di testo il cui nome viene richiesto in fase di elaborazione.

Tip fornito dal sig. C.Calabrò

```
#include <fstream.h>
#include <stdlib.h>
#include <string>
 char filename[255];
 string buffer1, buffer2;
 int n_lettere, scelta;
 fstream orig;
 fstream dest;
 void crypto(), decrypto();
void main() {
 cout<<"Benveuto in Crypto, di Cristiano Calabro'"<<endl;
 cout<<"Vuoi: \t1 => criptare \n\t2 => decriptare"<<endl;
```

```
cout<<"\n\tScelta => ";
cin>>scelta;
switch(scelta)
{ case 1:
  crypto(); break;
 case 2:
  decrypto(); break;}
system("PAUSE"); }
void crypto()
{ cout<<"\n\nInserisci il nome del file da aprire (comprensivo di
                                              estensione) => ";
 cout<<"\n\tNome del file => ";
 cin>>filename;
 orig.open(filename, ios::in);
 dest.open("crypted.dat", ios::out);
 if (!orig)
   cout<<"Errore nell'apertura del file"<<endl;
  else
  { orig.seekg(ios::beg); //posiziono il puntatore all'inizio del file
     while(!orig.eof())
   { orig>>buffer1; //carico in buffer la prima parola
      n_lettere=buffer1.size(); //n_lettere é un intero che contiene la
                                         quantità di lettere della parola
      n_lettere=buffer1.size(); //n_lettere é un intero che contiene la
                                         quantità di lettere della parola
 /* Segue il cuore del prgramma */
    for (int i=n_lettere-1; i>=0; i—) //mi muovo di carattere in
                                                    carattere a ritroso
    { buffer2[(n_lettere-1)-i]=buffer1[i]+n_lettere; //Scrivo in buffer2 il
                     caratteri traslati di n_lettere e ruotati di n_lettere_posizioni
      dest<<buffer2[(n_lettere-1)-i]; }
    dest<<" "; };
/* Termine del cuore del programma */
  cout<<"File criptato con successo!"<<endl; }
orig.close();
dest.close();}
void decrypto()
{ orig.open("crypted.dat", ios::in);
 cout<<"Inserisci il nome del file sul quale scrivere (sara' di tipo
                                              tuonome.txt) => ";
 cout<<"\n\tNome del file => ";
 cin>>filename;
 strcat(filename,".txt");
 dest.open(filename, ios::out);
 if (!orig)
   cout<<"Errore nell'apertura del file"<<endl;
   else
   { orig.seekg(ios::beg); //posiziono il puntatore all'inizio del file
     do
    { orig>>buffer1; //carico in buffer la prima parola  n_lettere=
       buffer1.size(); //n_lettere é un intero che contiene la quantità
                                                  di lettere della parola
       n_lettere=buffer1.size(); //n_lettere é un intero che contiene
                                            la quantità di lettere della parola
 /* Segue il cuore del prgramma */
    for (int i=n_lettere-1; i>=0; i—) //mi muovo di carattere in
                                                    carattere a ritroso
    { buffer2[(n_lettere-1)-i]=buffer1[i]-n_lettere; //Scrivo in buffer2
          il caratteri traslati di n_lettere e ruotati di n_lettere_posizioni
      dest<<buffer2[(n_lettere-1)-i]; }
     dest<<" ";
     }while(!orig.eof());
/* Termine del cuore del programma */
   cout<<"File decriptato con successo!"<<endl; }
 orig.close();
 dest.close();
}
```

Delphi

ENUMERARE LE CHIAVI DEL REGISTRO DI SISTEMA

Il tip consente di enumerare tutte le chiavi di registro modificate in un determinato intervallo di tempo.
È utile per vedere le modifiche effettuate sul registro di Windows da programmi installati (eventuale spyware allegato a software p2p, etc...). Il codice, data la sua prolissità, è presente nel CD-Rom allegato alla rivista e/o sul sito web di ioProgrammo (*www.ioprogrammo.it*).

Tip fornito dal sig. C.Pasolini

Cronaca di un attacco

Microsoft scivola su ASN.1

ASN.1 è una tecnologia che ha causato qualche imbarazzo e non pochi problemi a Microsoft. Si è temuto un nuovo worm di massa "Blaster-like", fortunatamente non è stato così.

C'è stato un periodo, relativamente recente, in cui la sigla "ASN.1" spuntava a sorpresa tra le righe e i popup di tutti i portali di sicurezza e di news tecnologiche. "Grave falla in una libreria di Windows" - "Microsoft rilascia l'ennesima patch per una vulnerabilità critica" - "Sistemi Windows a rischio per un nuovo attacco informatico"… questo era ciò che sbandieravano i portali IT (con il solito eccesso di enfasi che contraddistingue questi eventi) nel mese di marzo, senza però mai spiegare a fondo il significato di questo termine oscuro. Non poteva quindi mancare, nella nostra consueta rubrica dedicata agli exploit, un appuntamento con questo bug tanto discusso, in cui sono approfonditi alcuni aspetti di *ASN.1,* mostrando un exploit DoS basato su questo bug.

ASN.1, IL PADRE DI TUTTI I PROTOCOLLI

Chi definisce *ASN.1* come un "protocollo" incorre in un grave errore concettuale. *ASN.1* (*Abstract Syntax Notation One)* non è un protocollo come invece si potrebbe dire per il TCP/IP o per HTTP. *ASN.1* è qualcosa di molto più generico e astratto di un protocollo, è infatti una potente tecnologia utilizzata per definire un modo comune di scambio delle informazioni in sistemi eterogenei. In sostanza è un linguaggio usato per scrivere nuovi standard di comunicazione fra sistemi. Usando un formalismo più tecnico possiamo definire *ASN.1* come un "data decription language", cioè un linguaggio usato per descrivere la struttura dei dati trasferiti tra Application Layer e Presentation Layer nel modello *OSI (Open System Interconnection).*

Con *ASN.1* è possibile definire che cosa è un "tipo di dati", come è fatto un tipo di dati (ad esempio INTEGER, BOOLEAN, ecc.), quali sono le keyword di un linguaggio e cosa significano (ad esempio BEGIN, END, IMPORT, EXPORT, ecc.).

ENCODING RULES

Considerare *ASN.1* come un linguaggio di programmazione (ad esempio il C) non è una cosa che dovrebbe meravigliarci. La similitudine non è azzardata, perché se riflettiamo un attimo, il "linguaggio C" non è altro che un insieme di formalismi e di regole che permettono di descrivere un algoritmo teorico con una serie di istruzioni più o meno complesse. *ASN.1* allo stesso modo è un linguaggio che permette di descrivere il formato e le specifiche di un formato di scambio di dati. Tutti noi programmatori sappiamo bene che un listato scritto in un linguaggio high-level, deve essere prima interpretato e poi tradotto nelle istruzioni macchina specifiche (x86 nel nostro caso) che lo rendono eseguibile. Allo stesso modo le regole create secondo *ASN.1*, devono essere successivamente tradotte e compilate usando uno dei seguenti metodi, prima del loro utilizzo:

BER - Basic Encoding Rules
CER - Canonical Encoding Rules
DER - Distinguished Encoding Rules
PER - Packing Encoding Rules

Ogni metodo di codifica ha un proprio set di regole che consentono di convertire un dato *ASN.1* in una sequenza di byte che può essere trasferita su un qualsiasi canale di comunicazione. BER è sicuramente il metodo più comunemente usato e del quale esistono numerose implementazioni (è definito nelle ITU-T Recommendations X.209 e X.690). Prendendo proprio spunto da un articolo tecnico di Microsoft su ASN.1 (KB 252648), ecco un esempio derivato direttamente dall'implementazione delle direttive X.209. Consideriamo un generico tracciato record *"Personnel-Record"* relativo ai dati anagrafici di una persona (Tab. 1). La struttura che può descrivere in maniera generica il record in ogget-

Using ASN.1 (Abstract Syntax Notation 1): A Data Description Language
http://www.nal.usda.gov/pgdic/Probe/v2n2/using.html

ASN.1 : Articoli e Documentazione
http://www.cs.columbia.edu/~hgs/internet/asn.1.html

ASN.1 secondo Microsoft
http://support.microsoft.com/default.aspx?scid=kb;en-%20us;252648

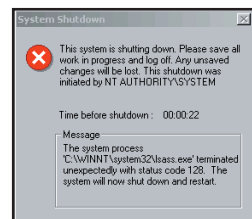



***Fig. 1:** Il crash causato dall'exploit per ASN.1 ricorda molto quello del worm Blaster: il processo di sistema LSASS va in tilt dopo la ricezione del pacchetto incriminato e obbliga il computer ad un riavvio forzato.*

[PersonnelRecord] #1	
Name:	John P Smith
Date of Birth:	17 July 1959
...(altri dati personali del record)	

TABELLA 1: n record con dati anagrafici.

```
PersonnelRecord ::= [APPLICATION 0] IMPLICIT SET {
        Name,
        title [0]          VisibleString,
        dateOfBirth [1]             Date,
        ... }
    Name ::= [APPLICATION 1] IMPLICIT SEQUENCE {
        givenName      VisibleString,
        initial        VisibleString,
        familyName     VisibleString
    }
```

TABELLA 2: Generalizzazione del record.

to può essere schematizzata come illustrato in Tab. 2. Questa descrizione è uno schema *"ASN.1 compliant"*, che specifica come i dati *"PersonnelRecord"* dovrebbero apparire e soprattutto come sono formattati, in modo del tutto astratto e generico (proprio per questo spesso ASN.1 viene accostato all'XML). Applicando il BER al dato ASN.1 così mappato, si ottiene una sequenza di dati nella forma di Tab. 3. Il pacchetto finale codificato che può essere trasmesso sul canale di comunicazione è quindi : *60 81 85 61 10 1A 04 "....", "...." 0A 43 08 19 59 07 17.*

Personnel							
Record	Length	Contents					
60	8185						
		Name	Length	Contents			
		61	10				
				VisibleString	Length	Contents	
				1A	04	"John"	
				VisibleString	Length	Contents	
				1A	01	"P"	
				VisibleString	Length	Contents	
				1A	05	"Smith"	
		DateofBirth		Length	Contents		
		A0		0A			
					Date	Length	Contents
					43	08	"19590717"

TABELLA 3: La sequenza di dati con BER.

NOTA

FIX SILENSIOSO?

Stando a quanto detto dagli esperti die eEye, questa vulnerabilità di *ASN.1* è stata silenziosamente fixata da Microsoft in Windows Server 2003 e Windows 2000 SP4; ciò è emerso da alcuni confronti fatti tra le diverse versioni della libreria *MSASN1.DLL* e la funzione *ASN1BERDecCheck()*.

GLI ADVISORIES

Gli advisory pubblicati sui problemi di ASN.1 nei sistemi Microsoft sono il bollettino di sicurezza MS04-07 e il documento tecnico redatto dal team di ricercatori di eEye. I link per rintracciare entrambi i documenti sono:

http://www.eeye.com/html/Research/Advisories/AD20040210.html
http://www.microsoft.com/technet/security/bulletin/MS04-007.mspx

Fig. 5: ***Il bollettino di sicurezza pubblicato sul sito di eEye. All'interno dell'advisory sono esplicitate tutte le funzioni a rischio esportate dalla libreria MSASN1.DLL.***

UN MONDO CHE GIRA SU ASN.1

Le potenzialità di *ASN.1* sono talmente vaste che oggi le sue applicazioni nel mondo delle telecomunicazioni ricadono praticamente ovunque: quasi tutti gli standard comunemente usati come *X.400 (Electronic Messaging), X.500 (Directory Services), X.200 (Network Communications), LDAP* e molti *RFC* sono basati su *ASN.1*. Appare quindi chiaro come Microsoft non abbia potuto fare a meno di implementare una propria libreria che nello specifico si concretizza nel file *MSASN1 .DLL* presente su quasi tutti i sistemi Windows. Ed è proprio questa libreria il punto dolente della questione *ASN.1:* nel mese di Febbraio i ricercatori di *eEye* hanno annunciato al mondo la scoperta di un heap overflow nelle funzioni di decodifica implementate da *MSASN1. DLL* e di conseguenza in tutti i prodotti ad essa correlati (la famiglia di sistemi operativi Windows NT/2000/XP, Internet Explorer, Outlook e il protocollo di autenticazione *NTLMv2*).

L'EXPLOIT DOS PER SISTEMI WINDOWS 2000 E XP

Il problema – secondo eEye – si concretizza fornendo un valore molto grande (range da *0xFFFFFFFD* a *0xFFFFFFFF*) nel campo "lunghezza" contenuto nel pacchetto *ASN.1*. Quando *MSASN1.DLL* cerca di decodificare un qualsiasi pacchetto, per prima cosa chiama la funzione *ASN1BERDecLength()* che ha lo scopo di acquisire la lunghezza effettiva di un valore e che successivamente passa la lunghezza calcolata alla funzione *ASN1BERDecCheck()*, per un ulteriore controllo di validità. Se al termine di tutti questi controlli non vengono riscontrate anomalie, la funzione che ha avviato la decodifica (ad esempio *ASN1BERDecOctetString()*) alloca una zona di memoria pari alla lunghezza determinata, arrotondata per eccesso al multiplo *DWORD* successivo. Quando viene passato uno un valore molto grande (come quelli suddetti), l'arrotondamento porta il valore a coincidere con zero, di conseguenza l'allocazione di memoria non avviene (viene allocata di fatto un'area con lunghezza zero) e le successive operazioni *memcpy()* causano l'overflow. Nell'advisory di *eEye* sono elencate di fatto tutte le API di *MSASN1.DLL* che potrebbero essere un veicolo per gli attacchi di overflow; scrivere un exploit di remote execution basato su questo bug è stato considerato arduo da tutti i ricercatori, a causa delle complesse interazioni tra la libreria *ASN.1* e le applicazioni. Tuttavia, un hacker di nome Christophe Devine è riuscito a creare un exploit di tipo Denial of Service, in grado di crashare da remoto i sistemi 2000 e XP. Il veicolo utilizzato per arrivare all'overflow di *MSASN1.DLL* è stato il servizio di sistema *LSASS.EXE* che resta in ascolto sulle porte 139 e 445 per ricevere i pacchetti di autenticazione del protocollo NTLMv2 (codificati proprio attraverso *ASN.1*). L'exploit in sé è banale: costruisce un pacchetto *NetBIOS* malformato e lo invia ad una delle due porte citate, causando l'overflow e il successivo crash di sistema, simile in tutto a quello del worm Blaster.

Il codice "incriminato", compilabile per sistemi Windows e Linux in base alla direttiva *#define WIN32*, lo trovate nel CD allegato alla rivista.

Elia Florio

Personalizzare la grafica delle applicazioni

Nuova "faccia" alle applicazioni!

Come rendere più flessibili e accattivanti i propri programmi con front-end che possono essere cambiate al volo. Il più famoso esempio di applicazioni con skin è Winamp…

Quando ci si affaccia nel mondo della programmazione di interfacce grafiche ci si imbatte inevitabilmente nell'uso in una serie di controlli standard offerti dal sistema operativo: bottoni, aree di testo editabile, scrollbar, combobox, etc.. La standardizzazione dei controlli, della forma delle finestre e persino della posizione delle voci di menu e degli shortcuts, oltre che una facilitazione per i programmatori, è anche una grande alleata degli utenti che, in questo modo, non si trovano a dover capire come utilizzare una funzione standard su ogni programma. Chiunque ha avuto esperienza con il lettore multimediale di Windows XP sa che, nella modalità denominata "*interfaccia*", non è immediato trovare persino il tasto di chiusura dell'applicazione.

Il rovescio della medaglia è che la standardizzazione delle interfacce grafiche porta ad una notevole piattezza delle applicazioni. Molti utenti amano modificare a proprio piacimento l'interfaccia dei programmi che usano o addirittura creare interfacce grafiche e condividerle con il popolo di Internet; non si spiegherebbe altrimenti il successo di siti come quello di Winamp; uno dei programmi che maggiormente incentiva la personalizzazione delle interfacce.

Di seguito andremo ad illustrare come creare un programma la cui interfaccia grafica può essere cambiata al volo, senza riavviare l'applicazione stessa. Vedremo come personalizzare anche i controlli (pulsanti etc..) e come evitare di dare ad ogni componente dell'applicazione la solita forma rettangolare, sfruttando il concetto di *region* offerto da Windows sin dalle sue prime versioni.

Prima di cominciare, vi ricordo che tutto il codice sorgente in C++/MFC(Visual Studio .NET) e i file di grafica ai quali si fa riferimento sono disponibili nel CD allegato nella cartella */soft/codice*.

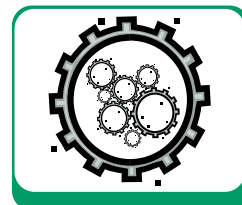
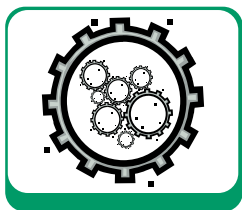
IL METODO CHE ADOTTEREMO

Nel panorama dei programmi che possiamo trovare in Rete e/o in commercio, esistono sostanzialmente due tipi di applicazioni la cui interfaccia è personalizzabile:

- Applicazioni il cui sfondo e i controlli possono essere personalizzati sostituendone le bitmap. Nelle quali la forma dell'applicazione resta, comunque, rettangolare e i controlli non possono essere spostati a proprio piacimento.

- Applicazioni il cui sfondo e i controlli possono essere personalizzati sostituendone le bitmap e specificando delle bitmap di *"maschera"* che ritagliano forme arbitrarie, non necessariamente rettangolari. Nelle quali esiste, tra l'altro, la possibilità di spostare e ridimensionare i controlli a proprio piacimento all'interno della finestra.

In questa sede tratteremo il secondo tipo di implementazione, per la quale dovremo prima stabilire un metodo di definizione del layout delle interfacce.

Creeremo, a tale scopo, un'applicazione in grado di caricare delle interfacce e definiremo due interfacce di test complete e contenenti: tre pulsanti, uno sfondo non rettangolare, uno scroller e un'area di testo che useremo per l'output. Per definire una interfaccia, avremo bisogno prima di tutto di una bitmap di sfondo che, per convenzione, sarà nera (RGB 0,0,0) nei punti che vogliamo "bucare" in modo da evitare l'uso di una ulteriore bitmap di maschera, e che avrà dei colori stabiliti laddove vogliamo inserire i controlli. Come possiamo vedere nella Fig. 1 è stata realizzata una interfaccia a forma di CD di ioProgrammo, la cui parte nera, una volta avviata l'applicazio-

Sebbene la potenza dei calcolatori è giunta a livelli sorprendenti, è sempre consigliabile attuare delle ottimizzazioni che evitino di dare l'impressione agli utenti che vi siano momenti in cui l'applicazione non risponde perché è impegnata in altro. Qui abbiamo, per esempio, ottimizzato la routine *GetPixel* per accelerare il processo di caricamento delle interfacce.

ne, non verrà mostrata, rendendo la nostra finestra tondeggiante. Le parti con colori uniformi rappresentano, invece, i segnaposto per i controlli. Come associamo però un controllo ad un determinato colore? A tale scopo, insieme all'interfaccia forniremo un file di testo con estensione *.skl* al cui interno adotteremo una semplice sintassi come quella che segue:

```
[Skin Layout]
<BackGround>      BlueBackGround.bmp
<Key>        QuitButton.bmp  0,255,255    "QUIT"
<Key>        Button1.bmp     255,0,0      "BUTTON1"
<Key>        Button2.bmp     255,255,0    "BUTTON2"
<scroller>   Scroller.bmp    0,255,0      "SCROLLER"
<textArea>   TextArea.bmp    161,231,69
```

La prima riga è utile solo a indicare che si tratta effettivamente di un file di layout per una interfaccia, in questa maniera eviteremo di caricare per errore file non corretti.

Le altre righe presentano in testa una parola chiave che identifica il tipo di dato che si sta per definire; ecco quindi: *<BackGround>* per lo sfondo della finestra, *<Key>* per i pulsanti, e gli ovvi *<scroller>* e *<textArea>* per i rispettivi controlli.

A seguire troviamo: il nome del file bitmap associato, i valori RGB che rappresentano il colore usato come segnaposto per un determinato controllo e l'identificativo che verrà usato all'interno del programma per riconoscere i controlli che, come vedremo fra poco, lanciano dei messaggi.

A titolo d'esempio, prendiamo in considerazione la terza riga di questo file di layout dove viene definito un bottone, la cui bitmap è *"QuitButton.bmp"*, che deve posizionarsi laddove troverà il primo pixel di colore 0,255,255 e che nel programma verrà associato all'identificativo *"QUIT"*. Riferendosi alla Fig. 1, il bottone appena specificato andrà a coprire la *X* celeste in alto e verrà usato come tasto di chiusura dell'applicazione.

Fig. 1: La bitmap di sfondo di una delle due interfacce create per il programma.

Sempre in riferimento alla Fig. 1 i segnaposto colorati hanno la forma dei controlli che li sovrascriveranno, questo accorgimento non è, in realtà, necessario; il programma usa quel colore per calcolare un rettangolo sul quale appoggiare il controllo. L'uso di un segnaposto con la forma corretta, tuttavia, semplifica l'interpretazione dell'interfaccia in fase di progettazione.

In Fig. 2 troviamo un esempio di pulsante, in questo caso proprio il tasto di chiusura del quale abbiamo appena parlato. Come si può notare la bitmap del bottone presenta una striscia verticale contenente tre fotogrammi dello stesso pulsante, questi tre fotogrammi rappresentano gli stati in cui si può trovare un bottone: non premuto, *highlight* (con il puntatore del mouse sopra), e premuto.

Fig. 2: La bitmap che rappresenta tutte le possibili posizione highlight e non di uno scroller.

In Fig. 3, possiamo vedere un esempio di bitmap per lo scroller che presenta diversi fotogrammi, uno per ogni posizione in cui può trovarsi lo scroller più le stesse posizioni in versione *highlight*.

Fig. 3: La bitmap che rappresenta tutti gli "stati" del bottone di chiusura dell'applicazione.

Come abbiamo accennato prima, la dimensione dei singoli fotogrammi dei controlli viene calcolata automaticamente dal programma grazie alla presenza del colore segnaposto sull'immagine di sfondo. Abbiamo visto come funziona la logica dell'applicazione che vogliamo sviluppare, vediamo adesso i dettagli implementativi di quanto discusso sino ad ora.

NOTA

Nel gergo informatico le interfacce personalizzate vengono chiamate skins esistono vere e proprie sezioni dedicate alle skins sui siti di parecchi software. Quello della personalizzazione delle interfacce dei programmi è un fenomeno simile alla personalizzazione dei cover per i cellulari non troppo datati. Le skin possono però aggiungere nuove funzionalità all'applicazione qualora fosse previsto un sistema di scripting.

L'APPLICAZIONE

L'applicazione è stata realizzata, con l'ausilio dal wizard di Visual Studio .NET, creando un semplice progetto *Single Document* (SDI) senza architettura document/view, senza system menu e senza alcun tasto di ingrandimento/riduzione della finestra.

A partire dal progetto vuoto si è provveduto alla rimozione della *childwindow* che MFC crea all'interno della finestra principale per ricoprire l'area client e alla rimozione della toolbar e del menu generati automaticamente Infine ci siamo occupati della rimozione delle risorse non più necessarie.

Nella *InitInstance* della classe *applicazione* è stato poi sostituito il codice di creazione della finestra principale con l'override del metodo create.

La classe della finestra principale è la responsabile del caricamento dell'interfaccia e della creazione dei controlli tramite il metodo *LoadSkin* al quale passeremo come parametro il nome del file con estensione *.skl*. *LoadSkin* distrugge, prima di tutto, i controlli già esistenti, in modo da poter ricaricare una interfaccia nel caso ve ne sia una già caricata, a tal fine si serve del metodo *DestroyAllControls*. Tutti i puntatori ai controlli creati sulla finestra vengono memorizzati in un vettore *stl(Standard Template Library)* di puntatori alla classe base dei controlli, basterà, quindi, scandire questo vettore per distruggerli.
Viene poi aperto il file di layout, verificato che sia un file di interfaccia valido e, a seconda dei vari tag incontrati durante la lettura, viene impostato lo sfondo e/o creati i controlli. La routine responsabile del caricamento dello sfondo è *LoadBackground*.
Per la memorizzazione e il caricamento dell'immagine di sfondo e delle immagini che rappresentano i controlli, ci affideremo all'oggetto *CImageEx* che abbiamo appositamente derivato dalla nuova classe *CImage* messaci a disposizione dalla nuova versione di MFC. Oltre alle funzionalità di caricamento dell'oggetto base, CImageEx offre un metodo *GetPixelEx* notevolmente più veloce di quello offerto dalla sua classe base, questo metodo necessità però di operare su una bitmap a 32 bit per pixel. Tutte le bitmap caricate verranno, quindi, convertite automaticamente in questo formato direttamente dal metodo *Load* di *CImageEx*. Per la conversione viene creata una nuova immagine a 32bpp e gli viene sovrapposta l'immagine da convertire che verrà successivamente distrutta. In questo modo vengono sfruttate le capacità di conversione automatica delle immagini contenute negli oggetti *CImage*. Una volta caricata l'immagine di sfondo deve essere ridimensionata la finestra dell'applicazione alle dimensioni della bitmap tramite il metodo *SetWindowPos* di *CWnd* e, come avevamo anticipato, devono essere "bucate" le parti colorate di nero, deve essere cioè possibile vedere attraverso questi ritagli.
Il meccanismo implementato in Windows per la creazione di finestre non rettangolari non è esattamente quello che viene definito uno strumento flessibile. Per bucare o ritagliare una finestra bisogna creare uno oggetto *CRgn* e poi assegnarlo alla finestra stessa tramite il metodo *SetWindowRgn* della classe *CWnd*. Vediamo in dettaglio come funziona questo processo. Un oggetto *CRgn* incapsula il concetto di *region* il quale, nell'interfaccia grafica di Windows, rappresenta un'area usata per varie funzionalità di clipping. Purtroppo però, mentre è assolutamente banale creare una region poligonale, ellittica o rettangolare, rispettivamente tramite i metodi *CreatePolygonRgn*, *CreateEllipticRgn* e *CreateRectRgn*, non si può dire altrettanto per una region di forma arbitraria, tutto quello che viene messo a disposizione dal sistema operativo è la possibilità di creare una region come unione di tanti rettangoli di dimensione qualunque. Creeremo le nostre region personalizzate attraverso l'unione di tanti piccoli rettangoli alti un pixel che seguiranno accuratamente la forma delle bitmap specificate come sfondo della finestra o dei controlli, tramite la funzione appositamente creata da noi: *CreateWndRegion*. In *CreateWndRegion* scandiamo la bitmap e testiamo ogni singolo pixel con il metodo *GetPixelEx* di cui abbiamo parlato prima per creare i rettangoli che coprono tutti i pixel dell'immagine che sono diversi da nero. È da notare il fatto che se avessimo eseguito questa operazione con il metodo *GetPixel* della classe base *(CImage)* il caricamento dell'interfaccia sarebbe stato notevolmente più lento, questo a causa della genericità di un metodo che deve essere in grado di funzionare con qualsiasi formato di immagine. Provare per credere. I rettangoli creati devono essere posti, insieme ad un header che esamineremo a breve, in uno spazio di memoria contiguo il cui puntatore deve essere passato come parametro al metodo *CreateFromData* di *CRgn*. L'header di cui abbiamo parlato è la struttura *RGNDATAHEADER* di Windows che deve essere riempita come segue:

```
Rdh->dwSize =  sizeof(RGNDATAHEADER);
Rdh->iType   =  RDH_RECTANGLES;

Rdh->nRgnSize  = NULL;
Rdh->rcBound    = areaRect;
```

Il campo *rcBound* deve specificare il rettangolo minimo che contiene tutta la region che creeremo, questo dato può essere ottenuto direttamente dalle dimensioni dell'immagine di sfondo:

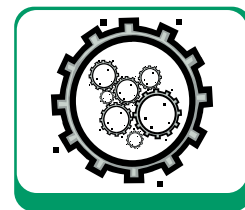
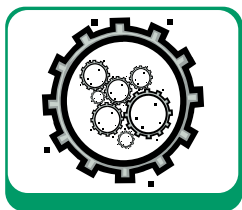
```
areaRect.SetRect(0, 0, pMaskImage->GetWidth(),
                       pMaskImage->GetHeight());
```

Per allocare lo spazio contiguo per i rettangoli e per la struttura header procederemo come segue:

```
unsigned long *RegionData=(unsigned long*)new
        unsigned char[sizeof(RECT) * Rectangles +
                        sizeof(RGNDATAHEADER)];
```

Dove *Rectangles* è una costante che rappresenta il numero massimo di rettangoli che si possono creare. E otterremo i puntatori all'header e al primo rettangolo in questa maniera:

```
RGNDATAHEADER *Rdh=(RGNDATAHEADER *)RegionData
RECT *Rect=(RECT *)(RegionData+(sizeof(
          RGNDATAHEADER)/sizeof(unsigned long)))
```

Nel codice abbiamo considerato il numero massimo di rettangoli che possono essere generati durante la

L'applicazione che abbiamo esaminato sfrutta un oggetto *CImageEx* direttamente derivato dalla classe *CImage* di MFC con la quale è possibile caricare i formati grafici *jpeg, bmp, gif* e *ping*, tutti i file di immagine delle interfacce potranno quindi essere forniti in questi formati.

creazione della region, al fine di accelerare al massimo questo processo. A questo punto è sufficiente dimensionare opportunamente i rettangoli e riempire la struttura per poi chiamare il metodo di creazione della region e associare quest'ultima alla finestra da bucare. Per la creazione dei controlli *LoadSkin*, dopo aver letto il file di layout al fine di ottenere il colore segnaposto, chiama il metodo *FindColorPos* il quale restituisce un rettangolo che racchiude tutte le occorrenze del colore specificato nell'immagine di sfondo. Siamo adesso in grado di disegnare lo sfondo della nostra finestra; di questo compito si occuperà il metodo *OnPaint* chiamando direttamente il metodo *Draw* dell'immagine caricata:

```
m_BgImage.Draw(dc.GetSafeHdc(), 0, 0)
```

L'immagine così disegnata copre perfettamente tutta l'area della finestra, di conseguenza non è necessario che la finestra venga cancellata prima di essere ridisegnata, per evitare fastidiosi problemi di flickering creeremo, allora, un override del metodo *OnEraseBackground* che lasceremo vuoto:

```
BOOL CSkinWnd::OnEraseBkgnd(CDC* pDC) {
                                        return TRUE; }
```

Cliccando con il tasto destro del mouse, tramite la funzione *TrackPopupMenu* verrà lanciato un menu popup dal quale si potrà decidere di caricare una nuova interfaccia o uscire dal programma. Trascinando il tasto sinistro del mouse sarà, invece, possibile spostare la finestra.
I controlli di cui tanto abbiamo parlato si basano tutti su un'unica classe base; *CSkinCtrl*. Questa classe, nel pieno rispetto dei meccanismi di ereditarietà, implementa tutte le funzionalità comuni a ad ogni tipo di controllo che vogliamo creare. In particolare permette: di caricare l'immagine contenente i fotogrammi che rappresentano i vari stati del controllo, di settare il fotogramma corrente e implementa un sistema di eventi *OnMouseEnter* e *OnMouseLeave*, che vengono chiamati quando il puntatore del mouse entra o esce dall'area del controllo. L'implementazione di questo sistema si trova all'interno del metodo *OnMouseMove* il quale cattura l'input del mouse, lancia un evento *OnMouseEnter* e controlla se il puntatore esce fuori dall'area del controllo per generare un evento *OnMouseLeave* e rilasciare il mouse. Gli eventi *OnMouseEnter* e *OnMouseLeave* verranno usati nei controlli per settare lo stato di highlight quando il puntatore vi si trova sopra, questo gradevole effetto grafico, molto apprezzato dagli utenti, permette di individuare velocemente le zone con le quali è possibile interagire. Ogni singolo controllo che deriva da *CSkinCtrl* implementa le sue funzionalità specifiche; il bottone cambia fotogramma quando il mouse lo attraversa e mostra il fotogramma "premuto" se necessario, lo scroller implementa il cambio dei fotogrammi quando viene trascinato il mouse dopo che vi si è cliccato sopra e la textarea, infine, fornisce un metodo per l'output di una stringa. Nell'applicazione d'esempio che stiamo analizzando è necessario che su ogni interfaccia creata vi sia almeno un'area di testo, dal momento che verrà usata per il logging degli eventi sui controlli.
La comunicazione tra i controlli e l'applicazione è garantita attraverso l'invio di *"user defined messages"*, dei veri e propri messaggi, come quelli di Windows i cui scopi possono però essere definiti dall'utente. Per definire un messaggio di questo tipo bisogna dichiararlo a partire dalla costante *WM_USER*:

```
#define WM_SKINCTRLMSG (WM_USER + 1)
```

Esattamente come avviene per i normali messaggi, l'invio di quello appena definito deve essere eseguito tramite il metodo *SendMessage* della classe *CWnd*. Tale messaggio viene intercettato tramite il metodo *OnSkinCtrlMessage* direttamente collegato al *WM_MESSAGE* di Windows tramite la macro di MFC *ON_MESSAGE* e il controllo che lo ha generato viene riconosciuto attraverso un nome identificativo assegnatogli in fase di caricamento dell'interfaccia e passato come parametro del messaggio.
La comunicazione tra i controlli e la finestra che li contiene sarebbe potuta avvenire anche tramite una chiamata diretta di un metodo del genitore del controllo, che si può ottenere tramite il metodo *GetParent*.
È comunque sconsigliabile questo tipo di approccio, dal momento che legherebbe l'implementazione del controllo a quello della finestra che lo crea.
Non ci resta che provare a lanciare l'applicazione.

L'uso dell'oggetto CImage di MFC implica l'utilizzo di alcune funzioni della cosiddetta gdiplus, una libreria contenuta nella dll di sistema (Windows XP) gdiplus.dll. Nel caso in cui si voglia fare girare l'applicazione su sistemi operativi diversi da Windows XP è necessario fornire questa dll copiandola nella cartella dell'eseguibile oppure, se posta in un percorso differente, registrarla tramite regsvr32.exe.

CONCLUSIONI

Come vedrete avviando l'applicazione, il metodo che abbiamo usato, benché non fosse particolarmente complesso, garantisce un ottimo risultato e raggiunge perfettamente lo scopo che ci eravamo prefissi.
I controlli creati per la trattazione di questo argomento sono puramente dimostrativi e potevano essere creati anche a partire da classi di controlli di Windows già incapsulati in MFC in modo da sfruttarne le caratteristiche intrinseche e apportando un minimo di personalizzazione, tuttavia si è preferito dare questa impostazione al progetto al fine di rendere l'argomento più scorrevole.
Spero di avere stimolato la vostra fantasia, non mi rimane che salutarvi e augurarvi buona programmazione

Amedeo Margarese

Un rilevatore di posizione in real-time

Integrazione del GPS con telefoni GPRS

Dopo aver mostrato come rilevare la propria posizione attraverso un ricevitore GPS, in questo appuntamento impareremo a conoscere, in anticipo, ciò che si "incontrerà" lungo il proprio percorso.

Sarebbe bello sapere se sul percorso che stiamo per compiere incotreremo un bar, un ristorante, un'autofficina, un nuovo centro commerciale o un rifornimento di carburante... come fare? Basta avere un Server con una banca dati, che associ ad ogni evento da segnalare la relativa coordinata geografica. Percorrendo la strada inviamo le nostre coordinate al Server che elaborerà le informazioni restituendo informazioni supplementari. Ma come inviare le coordinate al Server? Semplice! Basta un telefonino dotato di protocollo GPRS.
Con il GPRS è possibile trasferire file, interrogare database, sviluppare applicazioni remote, etc; per usufruire di tale tecnologia abbiamo bisogno di:

- un telefonino ed un contratto telefonico che supporti la tecnologia GPRS;
- l'abilitazione all'utilizzo del sistema GPRS;
- la conoscenza necessaria per la configurazione hardware e software del proprio cellulare.

IL SISTEMA DI COMUNICAZIONE CLIENT/SERVER

Il sistema che andremo a realizzare avrà, come è facile intuire, una struttura Client/Server; saranno quindi creati due diversi applicativi: *Client* e *Server*.

- Il *Client*, che risiederà sul nostro Notebook, avrà il compito di ricevere i dati della nostra posizione dal ricevitore GPS per poi inviarli al *Server*;
- Il *Server*, che risiederà su un computer remoto, avrà il compito di ricevere i dati dal *Client*, elaborarli e inviare il risultato dell'elaborazione nuovamente al *Client*.

Nei precedenti articoli abbiamo già visto come implementare un sistema per la ricezione dei dati da

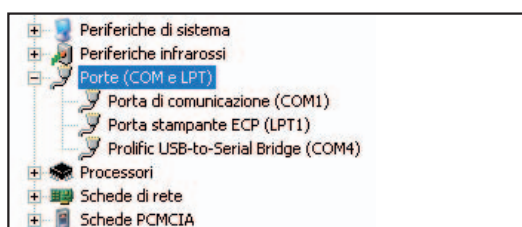



Fig. 1: ***Il processo per utilizzare la Dll NmeaParser.***

un sistema GPS collegato ad un Notebook. Utilizzeremo, quindi, la libreria sviluppata in precedenza, *NmeaParserDll*, per interfacciare l'applicativo alla porta USB cui è collegato il navigatore. *NmeaParserDll* riceve, tramite l'oggetto *MSComm*, stringhe di dati dalla porta seriale pur essendo il GPS collegato alla porta USB; questo è reso possibile poiché, durante la configurazione del navigatore il software fornito dalla casa madre crea una porta *Profilic USB-to-Serial Bridge* (Fig. 1). La *Dll* va innanzitutto linkata nei riferimenti dell'editor (Fig. 2).

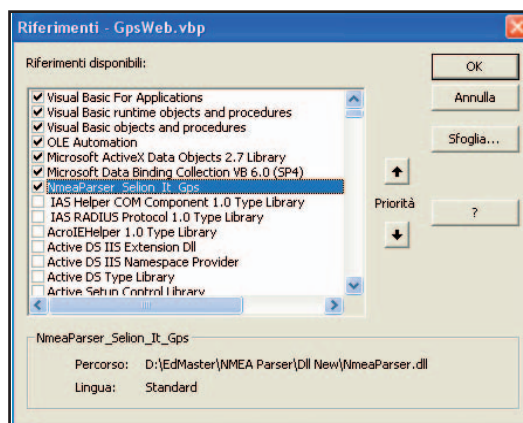
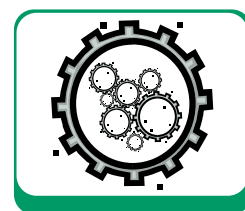
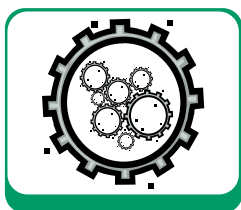



Fig. 2: ***Dal menu Progetto dell'IDE di Visual Basic 6 scegliamo la voce Riferimenti e quindi linkiamo la libreria opportuna.***

Inserita la DLL nei riferimenti, dovrà essere creata un'istanza dell'oggetto in esame:

```
Private objParser As New NmeaParser
```

GPRS

GPRS *(General Packet Radio Service)* è un servizio a valore aggiunto che permette di inviare e ricevere informazioni attraverso un network telefonico. La novità rispetto ai tradizionali sistemi per la trasmissione di voce è che nel GPRS non si parla più di commutazione di circuito ma di commutazione di pacchetto, utilizzando la stessa logica per la trasmissione dati su Internet. La massima velocità teorica consentita si aggira attorno ai 171 Kb/s. Il GPRS sfrutta la modalità di connessione *"always on"*, ovvero si rimane sempre collegati alla rete come se si disponesse di una linea dedicata.

Una volta invocata la funzione *ProcessData,* tutte le informazioni fornite dal Navigatore potranno essere acquisite nelle variabili illustrate in Tab. 1.
Esempio: *txtAltSea.Text = objParser.GGA_ALTITUDE txtLatitude.Text = objParser.GGA_LATITUDE*

L'APPLICATIVO LATO CLIENT

L'applicativo lato *Client* si presenta così come mostrato in Fig. 3. Sulla parte sinistra, in alto, sono ben visibili le coordinate geografiche (latitudine, longitudine e relative direzioni), l'altitudine e la velocità, tutti dati ricevuti dal GPS.
In basso i *Frames* che servono a monitorare lo stato del ricevitore GPS e quello della connessione del *Server*. Iniziamo quindi ad analizzare il codice che viene eseguito al click del pulsante *Start*:

Variabile	Descrizione
GPS_STATE	Indica lo stato del GPS, se riceve o meno dati dal satellite
RMC_PROCESS_STATE	Segnala errori nella trasmissione RMC
GGA_PROCESS_STATE	Segnala errori nella trasmissione GGA
RMC_PREF	Prefisso della frase RMC
RMC_TIME	Ora attuale
RMC_STATUS	Stato della trasmissione
RMC_LATITUDE	Latitudine attuale
RMC_N_S	Direzione della latitudine
RMC_LONGITUDE	Longitudine attuale
RMC_E_W	Direzione della longitudine
RMC_SPEED	Velocità al suolo
RMC_DATE	Data attuale
RMC_CHK_SUM	Cheksum della frase RMC
GGA_PREF	Prefisso della frase GGA
GGA_TIME	Ora attuale
GGA_LATITUDE	Latitudine
GGA_N_S	Direzione della latitudine
GGA_LONGITUDE	Longitudine

TABELLA 1: Le informazioni rese disponibili dal navigatore.

```
Private Sub bStart_Click()
  If bStart.Caption = "Start" Then
    bStart.Caption = "Stop"
    MSComm1.CommPort = 4
    MSComm1.Settings = "4800,N,8,1"
    MSComm1.InputLen = 0
    MSComm1.PortOpen = True
    'Iniziano le connessioni al server
    TimerServer.Enabled = True
    TimerGPS.Enabled = True
    lGpsStatus.Caption = "In Collegamento..."
  Else
    bStart.Caption = "Start"
    List1.Clear
    TimerGPS.Enabled = False
    Clear
    If MSComm1.PortOpen = True Then
      MSComm1.PortOpen = False
    End If
    StatusGPS = False
    lGpsStatus.Caption = "Dispositivo GPS non
                                    connesso..."
    lServerStatus.Caption = "Server non connesso..."
    tcpClient.Close
  End If
  Exit Sub
End Sub
```

La prima operazione che viene svolta consiste nel controllare se la proprietà *Caption* del controllo *bStart* è impostata sul valore *"Start"*, se tale condizione è veritiera significa che l'utente vuole "avviare" l'applicativo, in questo caso sarà eseguita la configurazione dell'oggetto *MsComm* (saranno ovvero assegnati i valori alle proprietà *CommPort, Settings* e *InputLen* del controllo); successivamente, impostando la proprietà *OpenPort* a *True* sarà "aperta" la porta di comunicazione che gestisce il GPS. Saranno quindi avviati i Timer *TimerGPS* e *TimerServer;* allo scadere dell'intervallo di tempo prefissato (valore della proprietà *Interval*), *TimerGPS* esegue la funzione *Get_Process_Data*. È probabile che il *Server* non sia immediatamente disponibile, per cui si rende necessario un componente *Timer, (*in questo caso *TimerServer)* che, ad intervalli prefissati, tenterà di stabilire una connessione:

```
  CountTimerServer = CountTimerServer + 1
  lServerStatus.Caption = "In Collegamento... Tentativo
        di connessione al Server: " & CountTimerServer
  ConnectToServer txtIp.Text
```

Ogni qual volta sarà generato l'evento *Timer* del controllo *TimerServer* sarà incrementata la variabile globale *CountTimerServer,* in modo da tener traccia di quante volte si è tentata l'operazione di connessione al *Server* senza successo; imposteremo massimo 20 tentativi. Ritorniamo all'evento *click* del pulsante *bstart:* se la proprietà *Caption* del controllo è diversa da *"Start"* è evidente che l'utente vorrà arrestare l'esecuzione del programma, non volendo, di fatto, più ricevere dati dal satellite; l'operazione la si compie chiudendo la connessione con la ricevente GPS, e la connessione con il Server (sempre che la stessa sia stata attivata!). Di seguito la procedura che mostra come avviene la connessione con il Server:

```
Sub ConnectToServer(ByVal Ip As String)
  If tcpClient.State = 0 Then
    tcpClient.Connect Ip
  End If
  If tcpClient.State = 7 Then
    TimerServer.Enabled = False
    lServerStatus.Caption = "Connessione con il
                                   Server riuscita"
    CountTimerServer = 0
  End If
  If CountTimerServer = 20 Then
    lServerStatus.Caption = "Impossibile stabilire
                         una connessione con il Server"
    TimerServer.Enabled = False
  End If
End Sub
```

NOTA

VARIABILI DELLA DLL

DISTANCE

Distance è di tipo intero, serve a modificare l'unità di misura della velocità, i possibili valori sono: 0, nel qual caso l'unità di misura sarà espressa in Km/h; 1, nel caso in cui si voglia la velocità espressa in Miglia/h

VALIDATEK

ValidateK è di tipo stringa, impostando correttamente il valore, il componente non ha nessun limite di tempo.

Per stabilire una connessione tra l'applicativo che risiede sul nostro Notebook *(Client)* e quello che risiede su un computer remoto (*Server*), utilizzeremo il classico protocollo TCP *(Transfer Control Protocol)*. Stabilita la connessione, entrambi i computer possono sia ricevere che inviare dati. Per accedere ai sevizi di rete Visual Basic mette a disposizione il componente *Winsock*, che funziona sia con il protocollo TCP che UDP *(User Datagram Protocol)*. Attraverso il controllo *WinSock* è possibile realizzare applicativi di rete, senza necessariamente, conoscere, in maniera approfondita, i protocolli di comunicazione: è infatti il controllo stesso che mette a disposizione del programmatore eventi e proprietà.
Analizzando il codice poc'anzi proposto, notiamo che è fondamentale conoscere lo stato della connessione; in Tab. 2 sono elencati i diversi valori con relativa descrizione dell'interrogazione della proprietà *State* del componente *Winsock*. Se lo stato della connessione (visualizzato dalla proprietà *State* del controllo *Winsock)*, è pari a 0 (vedi Tab. 2) la connessione è chiusa. Se ci troviamo in questa situazione, dobbiamo collegarci al Server richiamando il metodo *Connect*, del controllo. Per richiamare il metodo *Connect* è necessario conoscere il nome o l'indirizzo del Server, nonché la porta di comunicazione. Nel nostro caso abbiamo scelto di rendere variabile l'indirizzo del Server facendolo scrivere nell'apposito TextBox, mantenendo fissa la porta di comunicazione. Quest'ultima può essere impostata all'interno della proprietà *LocalPort* di Winsock; il valore di default di *LocalPort* è 0, così da selezionare il numero della porta in maniera completamente casuale. Abbiamo visto, quindi, come effettuare la connessione con il Server. Interrogando nuovamente la proprietà *State*, sempre del controllo *Winsock*, se il risultato sarà 7 la, connessione sarà riuscita, in questo caso la prima operazione da compiere è avvisare l'utente che da questo momento il Server è disponibile a ricevere ed inviare dati, (frame *Stato Server*); successivamente stoppiamo il *TimerServer* settando a *False* la proprietà *Enabled:*
Vista tutta la fase di connessione con il Server analizziamo ora la funzione *Get_Process_Data*

```
Sub Get_Process_Data()
On Error GoTo error_handler
    objParser.Distance = 0
    objParser.ValidateK = "NMEADll_Selion.it"
    objParser.ProcessData CStr(MSComm1.Input)
    If objParser.GPS_STATE = False Then
        ...
        lGpsStatus.Caption = "Il dispositivo GPS è
            connesso, ma non riceve segnali dal satellite..."
    Else
        ...
        lGpsStatus.Caption = "Il dispositivo GPS è
                                        connesso..."
        'sckOpen 7 Aperto
        ...
        Else
        ...
    End If
    Exit Sub
error_handler:
    MsgBox Err.Number & ", " & Err.Description
    bStart = True 'Chiude tutto
End Sub
```

All'interno di questa funzione viene utilizzato l'oggetto *objParser*, bisogna quindi andare a rivedere quanto scritto sopra per utilizzare al meglio la Dll che interfaccia il GPS con l'applicativo. Impostiamo, dapprima, la proprietà *Distance* dell'oggetto *objParser* a 0, in modo da esprimere la velocità in chilometri orari, *ValidateK* è *"NMEADll_Selion.it"*, in modo da utilizzare senza nessun limite di tempo l'oggetto. Alla funzione *ProcessData*, sempre dell'oggetto *objParser*, viene passata la stringa restituita dalla proprietà *MSComm1.Input;* questo restituirà l'intero flusso di dati tra la porta di connessione ed il GPS.

Fig. 3: *L'applicativo client: sulla parte destra è visibile il pulsante Start per avviare il sistema di ricezione dati dal satellite e l'invio dei dati al Server.*

Costante	Valore	Descrizione
sckClosed	0	Impostazione predefinita. Chiuso
sckOpen	1	Aperto
sckListening	2	In attesa
sckConnectionPending	3	Connessione in sospeso
sckResolvingHost	4	Risoluzione dell'host in corso
sckHostResolved	5	Host risolto
sckConnecting	6	Connessione in corso
sckConnected	7	Connesso
sckClosing	8	Il client sta chiudendo la connessione
sckError	9	Errore

TABELLA 2: Le possibili impostazioni della proprietà State

A questo punto l'oggetto *objParser*, tramite la funzione *ProcessData*, ha elaborato la stringa ricevuta e inizializzato tutte le variabili che identificano la nostra posizione. Interrogando la proprietà *objParser.GPS_ STATE*, siamo ora in grado di stabilire se il GPS è in grado di ricevere o meno i dati dal satellite. Molti ricevitori satellitari hanno una luce rossa che, se lampeggiante, indica assenza di segnale, interrogando *objParser.GPS_STATE*, riceveremo valore *False* qualora il segnale satellitare sia assente o debole; viceversa, qualora *objParser.GPS_STATE* restituisca il valore *True*, saranno aggiornate le *TextBox* con i valori ricevuti dalla ricevente: velocità, latitudine,

longitudine, altitudine etc. A questo punto, viene valutato lo stato della connessione con il Server: se il Server è connesso, viene inviata una stringa di dati al Server, viceversa sarà segnalato all'utente che il Server non è momentaneamente disponibile, consigliando di attendere per riprovare a stabilire nuovamente la connessione. La ricezioni delle informazioni provenienti dal Server è opzionale, in quanto vi è un *CheckBox* sul Form che consente di Attivare/Disattivare la ricezione dei dati, questo è possibile poiché la stringa dati viene inviata a Server solo ed esclusivamente se il CheckBox è attivo. Disattivando la ricezione dati il programma fornirà solo i dati relativi alla nostra posizione. La stringa dati che il Client invia al Server ha la seguente struttura:

```
"Dat:" & txtSpeed.Text & "#" & txtLatitude.Text & "#"
    & txtLongitude.Text & "#" & txtAltSea.Text & "#" &
        txt_N_S.Text & "#" & txt_E_W.Text & "#" &
                                    txtDistance.Text,
```

Questa, come già detto in precedenza, sarà interpretata dal Server in modo da rispondere al Client, e dare quindi le informazioni relative alle elaborazioni del Server. La procedura che segue mostra come la ricezione dei dati dal Server da parte del client:

```
Private Sub tcpClient_DataArrival(ByVal bytesTotal As Long)
    Dim rcvbuf As String
    tcpClient.GetData rcvbuf
    Select Case Left(rcvbuf, 4)
        Case "Con:"
            SendDataToServer "AcceptConnect"
        Case "Dat:"
            ProcessData Mid(rcvbuf, 8, Len(rcvbuf)),
                                        Mid(rcvbuf, 5, 3)
        Case "Msg:"
            List1.AddItem Right(rcvbuf, Len(rcvbuf) - 4)
        Case Else
            List1.AddItem rcvbuf
    End Select
Exit Sub
Error:
    SendDataToServer "DoneTrans"
    tcpClient.Close
End Sub
```

L'evento *DataArrival* del componente *tcpClient* viene generato alla ricezione di nuovi dati da parte del computer in cui si è stabilita una connessione TCP. All'interno dell'evento indicato viene eseguita la porzione di codice appena descritta. Il valore della variabile *rcvbuf* è il blocco di dati ricevuto dal Server, recuperato attraverso il metodo *GetData* di *tcpClient*. Se il dato ricevuto (che è poi una stringa) ha i primi quattro caratteri uguali a *"Con:"* significa che la richiesta di connessione è stata accettata da parte del Server, viene quindi nuovamente inviata una stringa di dati al Server che verrà interpretata da quest'ultimo. Se invece il dato ricevuto dal Client ha i primi quattro caratteri uguali a *"Msg:"* significa che è arrivata la risposta del Server relativa alle informazioni richieste. Verrà quindi processata la stringa ricevuta attraverso la funzione *ProcessData*.

```
Sub ProcessData(StringData As String, NumField
                                            As String)
    Dim X
    Dim ARR_DATA(1 To 100) As String
    List1.Clear
For X = 1 To CInt(NumField)
   List1.AddItem "------------ " & Now() & " ------------"
   List1.AddItem X & ") " &  = sGetToken(StringData, X)
   List1.AddItem " "
Next X
End Sub
```

ProcessData si avvale di due argomenti: il primo è la stringa dati vera e propria, ossia i dati ricevuti dal Server in cui è stata esclusa la stringa *"Msg:"*, il secondo, invece, rappresenta il numero di informazioni che il Server è riuscito a recuperare. Ricevuta la stringa, bisogna solo "scompattarla" tramite la funzione *sGetToken*, i dati vengono così "ricopiati" nella relativa ListBox. È da sottolineare che l'applicativo riceve le informazioni dal Server in funzione della distanza inserita nella TexBox *txtDistance:* variando quest'ultima si potrà conoscere anticipatamente cosa incontreremo lungo il nostro tragitto. Naturalmente, il numero massimo di informazioni che è possibile ricevere non è infinito, ma più precisamente impostato a 30, questo per non appesantire le comunicazioni tra Client e Server.

UNA PRECISAZIONE SULLA DISTANZA TRA I PUNTI

La distanza tra i punti è rappresentata dalla misura in linea d'aria tra questi, non è quindi paragonabile alla distanza che viene percorsa con l'automobile.

LA STRUTTURA DELL'APPLICATIVO SERVER

Analizzeremo ora l'applicativo che sarà caricato sul Server. È impossibile interagire direttamente con l'applicativo (a meno che non si voglia chiudere il programma!), il Server ha il compito di soddisfare solo ed esclusivamente le richieste fatte del Client.

```
Private Sub Form_Load()
    tcpListen.Listen
End Sub
```

Non appena il Form viene caricato, automaticamente viene instaurata una connessione, ponendola in modalità di "ascolto". L'intera operazione è gestita dal componente di rete *Winsock*, utilizzando il metodo *Listen*. In presenza di una connessione viene generato l'evento *ConnectionRequest*. Durante la

gestione dell'evento *ConnectionRequest*, è consigliabile che l'applicazione utilizzi il metodo *Accept* in una nuova istanza del controllo per accettare la connessione, per questo motivo utilizziamo l'oggetto *tcpServer:*

```
Private Sub tcpListen_ConnectionRequest(ByVal
                                requestID As Long)
    If tcpServer.State <> sckClosed Then
                        tcpServer.Close: DoEvents
    tcpServer.Accept requestID
End Sub
```

Client e Server sono connessi: da questo momento, è possibile, inviare e ricevere dati all'interno del sistema. Il Server è in attesa di dati e si mette in ascolto sul canale stabilito per la connessione, nel momento si ricevono dei dati il controllo *tcpServer* invoca l'evento *DataArrival:*

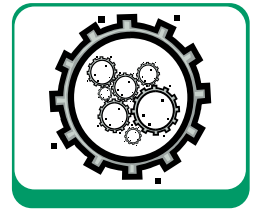
```
Private Sub tcpServer_DataArrival(ByVal bytes3Total
                                        As Long)
    Dim rcvStr As String
    tcpServer.GetData rcvStr
    Select Case Left(rcvStr, 4)
        Case "Con:"
            SendDataToClient "AcceptConnect"
        Case "Dat:"
            ServerAnswer (Mid(rcvStr, 5, Len(rcvStr)))
        Case Else
            SendDataToClient "Messaggio Incomprensibile"
    End Select
    List1.AddItem rcvStr
End Sub
```

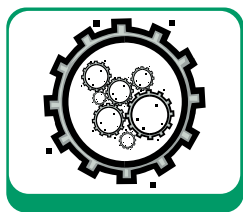
Il metodo *GetData* recupera i dati inviati dal Client, memorizzandoli nella variabile *rcvStr.* A questo punto va analizzato il messaggio ricevuto. Vengono valutati i primi quattro caratteri della stringa, se la stringa inizia con *"Con:"*, viene inviato al Client un messaggio che indica che la connessione è stata accettata, viceversa se la stringa inizia con *"Dat:"* il sistema indicherà che il Client ha inviato la sua posizione geografica e attende informazioni dal server.

```
Sub ServerAnswer(StrData As String)
    Dim x
    Dim ARR_DATA(1 To 7) As String
    For x = 1 To 7
        ARR_DATA(x) = sGetToken(StrData, x)
    Next x
    For x = 1 To 7
        List1.AddItem ARR_DATA(x)
    Next x
    CheckData ARR_DATA(1), CDbl(ARR_DATA(2)),
      CDbl(ARR_DATA(3)), ARR_DATA(4), ARR_DATA(5),
                    ARR_DATA(6), CDbl(ARR_DATA(7))
End Sub
```

La stringa di dati ricevuta sarà passata alla funzione *ServerAnswer* che ha il compito, innanzitutto, di "Scompattare" il messaggio del Client (attraverso la funzione *sGetToken*) per poi passare i singoli dati alla funzione *CheckData.*

```
Sub CheckData(SpeedData As String, LatitudeData As
    Double, LongitudeData As Double, AltitudeData As
        String, NS_Data As String, EW_Data As String,
                              Dist_Data As Double)
    Dim strDBName, strConnect As String
    Dim SqlStr As String
    Dim VarCheckData As String
    strDBName = "Db.mdb"
    strConnect = "Provider=Microsoft.Jet.OLEDB.4.0;
                                     Data Source="
    strConnect = strConnect & App.Path & "\" &
                                        strDBName
    Set Cn = New ADODB.Connection
    Set Rs = New ADODB.Recordset
    Cn.Open strConnect
    Rs.CursorType = adUseClient
    Rs.LockType = adLockPessimistic
    Rs.Source = "SELECT * FROM Dati WHERE note<>'';"
    Rs.ActiveConnection = Cn
    Rs.Open
    VarCheckData = ""
    Dim Cont, FCont, Dist, MaxInfo
    MaxInfo = 30
    Cont = 0
    Do While Not Rs.EOF
     Dist = Distance(Rs.Fields("Latitude"),
                Rs.Fields("Longitude"), LatitudeData,
                                    LongitudeData)
     If Dist < Dist_Data Then
        If Cont < MaxInfo Then
         VarCheckData = VarCheckData & Rs.Fields(
         "Note") & ". Distanza: " & Round(Dist, 2) & "#"
         FCont = Cont + 1
        End If
        Cont = Cont + 1
     End If
     DoEvents
     Rs.MoveNext
    Loop
    Rs.Close
    Cn.Close
    Set Rs = Nothing
    Set Cn = Nothing
    If Cont < 10 Then
      SendDataToClient "Dat:00" & FCont & VarCheckData
    Else
      SendDataToClient "Dat:0" & FCont & VarCheckData
    End If
End Sub
```

Vengono passati tutti i dati ricevuti dal Client, quindi velocità di marcia, latitudine, longitudine, altitu-

GETDATA

Il metodo *GetData* recupera i dati inviati dal Client, memorizzandoli nella variabile *rcvStr*. Vengono valutati i primi quattro caratteri della stringa, se la stringa inizia con *"Con:"*, viene inviato al Client un messaggio che indica che la connessione è stata accettata, viceversa se la stringa inizia con *"Dat:"* il sistema indicherà che il Client ha inviato la sua posizione geografica e attende informazioni dal server.

dine, direzione della latitudine, direzione della longitudine e la distanza massima delle informazione che si desidera avere. L'applicativo lato Server può svolgere la propria funzione grazie ad un database relazionale che contiene le coordinate dell'evento da segnalare all'utente. *CheckData* dovrà connettersi al DBMS e selezionare le informazioni che il Client desidera ricevere. Come già precedentemente asserito, è stato inserito un limite al numero di informazione che il Server può inviare. Il Server risponderà al Client con una stringa che sarà interpretata e successivamente visualizzata sul Form dell'utente. Scorrendo il database, calcoliamo la distanza tra il dato corrente nel DBMS e la nostra posizione, se questa è minore della distanza massima per ricevere informazioni viene memorizzato in una stringa il valore del campo Info del database. Il calcolo della distanza tra due punti attraverso le coordinate è stato fatto attraverso l'uso del codice acquisito presso il sito *Argray.org:*

```
// CONSTANTS USED INTERNALLY
final double DEGREES_TO_RADIANS = ( Math.PI/180.0 );
// Mean radius in KM
final double EARTH_RADIUS = 6371.0;
public double  GreatCircleDistance(LatLong alt)
{
  double p1 = Math.cos(lat)*Math.cos(lon)
                       *Math.cos(alt.lat)*Math.cos(alt.lon);
  double p2 = Math.cos(lat)*Math.sin(lon)
                       *Math.cos(alt.lat)*Math.sin(alt.lon);
  double p3 = Math.sin(lat)*Math.sin(alt.lat);
  return(Math.acos(p1+p2+p3)*EARTH_RADIUS);
}
```

Visti il funzionamento del Client e del Server, siamo in grado di utilizzare l'intero sistema: basta accendere il GPS, avviare l'applicativo *Client*, inserire nell'apposito campo l'indirizzo del Server e il gioco è presto fatto, riceveremo le informazioni richieste ogni dieci secondi. Ci resta da vedere solo come inserire nuovi dati all'interno del database remoto, in modo che quando il Client passa per quella determinata posizione possa ricevere l'informazione associata. All'uopo progettiamo un'applicazione denominata *ServerUpdate* che è molto simile al programma lato Client, questa deve essere in grado sia di identificare la posizione attuale che di inviarla al Server insieme alla descrizione dell'evento che dovrà essere associato a quel determinato luogo.

Fig. 5: ***Un esempio che chiarisce, graficamente, quando asserito: la linea in blu rappresenta il percorso automobilistico per andare dal punto 1 al punto 2, la linea in nero, invece, sta ad indicare la distanza più breve per andare da un punto all'altro.***

Fig. 6: ***L'applicazione che serve ad aggiornare le informazioni presenti sul server.***

COME FUNZIONA SERVERUPDATE

L'interfaccia grafica dell'applicazione *ServerUpdate* è molto simile all'applicazione Client: è presente il campo per l'inserimento dell'Ip del Server per la connessione tra gli applicativi e le TextBox contenenti i dati sulla latitudine, longitudine e altitudine. È stato inserito il campo *Dati* utile per inviare il messaggio che sarà visualizzato all'utente ("Centro commerciale Pippo", o ancora "Rifornimento carburante Pluto"). L'invio dei dati al Server è possibile solo ed esclusivamente quando è attiva una connessione con quest'ultimo quando, ovviamente, il GPS riceve dati dal satellite; la proprietà *Enabled* del pulsante *"Invia al Server"* è settata a *True* solo quando le due condizione sono verificate. Se *"Invia al Server"* è attivo avremo la possibilità di aggiornare il database remoto. *ServerUpdate* invierà al Server una stringa di dati contenenti le informazioni relative a: latitudine, longitudine, altitudine, direzione di latitudine e longitudine ed il messaggio da associare alle coordinate appena menzionate. Il server, come abbiamo già visto, effettua il parsing della frase ricevuta: se questa inizia per *"Upd:",* è accertato che le informazioni provengono dall'applicazione *ServerUpdate.* Il Server dovrà, quindi, analizzare la frase per poi estrarre i singoli campi da memorizzare in un nuovo record del DBMS.

Luigi Salerno

Web Service Enhancements 1.0 for .NET

Web Services: l'invio di file

Fin dalla presentazione di .NET alla PDC del 2000, è stata data grande importanza alla capacità del .NET Framewok, e di ASP.NET in particolare, di facilitare realizzazione e utilizzo di Web Service.

Le capacità di .NET sono subito apparse impressionanti, rispetto alle tecnologie allora esistenti: bastava mettere un attributo per trasformare un qualsiasi metodo in un Web Service, e bastava aggiungere un riferimento Web ad un progetto qualsiasi sviluppato col Visual Studio .NET per poterlo utilizzare. Svanito l'entusiasmo iniziale, ci si è accorti che la promessa dei Web Service era incompleta. Al protocollo SOAP usato per implementarli, mancavano una serie di funzionalità, tra cui la security, la possibilità di trasportare attachments binary, la gestione delle transazioni, la gestione degli eventi, ecc…

LIMITAZIONI DEI WEB SERVICE DI .NET

La security poteva essere gestita attraverso IIS, oppure programmaticamente, ma entrambi i modi limitavano l'interoperabilità. Per tutti le altre necessità invece non c'era nulla, con conseguenti sforzi da parte degli sviluppatori per implementare soluzioni custom, che andavano poi adattate alle varie situazioni. Per ovviare alle limitazioni Ibm, Bea e Microsoft (a proposito, come fatto notare da Don Box di Microsoft, avete mai fatto caso a cosa si ottiene prendendo la prima lettera di ogni società?) hanno iniziato a lavorare a una serie di protocolli modulari per implementare le funzionalità mancanti. Questi protocolli sono quelli della famiglia *WS-**, e sono ormai più di una decina. Microsoft ha scelto di renderli disponibili in .NET mano a mano che venivano rilasciati tramite un tool chiamato *Web Service Enhancements for .NET* (WSE), attualmente in versione 1.0 SP1. Questo tool è pienamente supportato da Microsoft e può essere scaricato gratuitamente. In questa prima versione sono stati implementati *WS-Security, WS-Routing* e *WS-Attachments*. È in corso di lavorazione la versione 2.0, che aggiunge altri protocolli alla suite iniziale, ma attualmente è in fase di technical preview, e non è supportata in contesti di produzione.

UTILIZZO DI WS-ATTACHMENTS

Vediamo in pratica come costruire un servizio per trasferire file attraverso SOAP e *WS-Attachments*. Questo servizio accetterà l'invio di 0 o più file binari, e di un eventuale messaggio di testo in chiaro, creerà una sottocartella per ogni ricezione e memorizzerà al suo interno ogni file ricevuto. Per semplificare l'utilizzo di WSE, Microsoft ha creato un tool chiamato

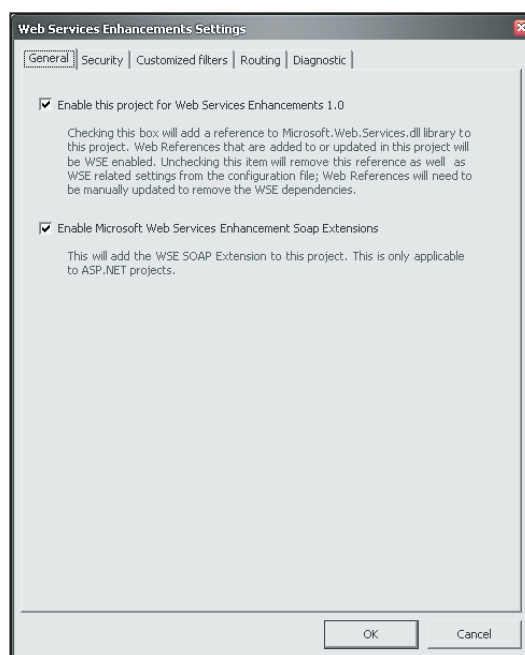
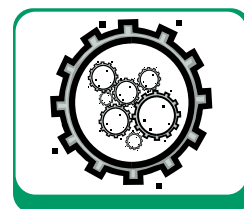
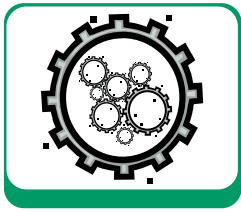
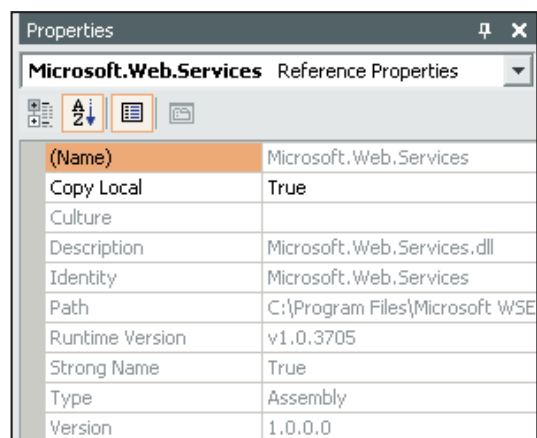



Fig. 1: ***Il pannello dove settare i parametri per la prima pagina.***

WS-ATTACHMENTS E DIME

WS-Attachments si basa su DIME per trasportare i file binari. DIME è una proposta di standard fatta da Microsoft. L'utilizzo di DIME non è compatibile con altri standard per l'invio di attachment binary, come ad esempio SWA, SOAP with Attachments.

WSE Settings Tool, che va installato prima di cominciare. Questo tool permette di automatizzare operazioni ripetitive come l'inserimento delle reference alla dll di WSE e la configurazione del *web.config*.
Iniziamo col creare il servizio che accetta i file e li salva sul disco. Dopo aver creato un progetto di tipo Web Service, bisogna configurare ASP.NET per riconoscere e gestire i nuovi protocolli. Bisogna andare nel *Solution Explorer*, selezionare il progetto appena creato, tasto destro del mouse, e selezionare *WSE Settings*. Apparirà un pannello dove bisogna abilitare entrambi i settaggi della prima pagina, come in Fig. 1.
Il tool modificherà il file *web.config* aggiungendo la registrazione del WSE all'interno di ASP.NET:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    ...
    <!— configurazione aggiunta dal WSE 1.0 Settings Tool—>
    <webServices>
      <soapExtensionTypes>
        <add type=
          "Microsoft.Web.Services.WebServicesExtension,
          Microsoft.Web.Services, Version=1.0.0.0,
          Culture=neutral,
          PublicKeyToken=31bf3856ad364e35"
          priority="1" group="0" />
      </soapExtensionTypes>
    </webServices>
    <!— fine configurazione aggiunta dal WSE 1.0
                                        Settings Tool —>
    ...
  </system.web>
</configuration>
```

Il tool aggiungerà poi al progetto il riferimento all'assembly *Microsoft.Web.Services.dll*. Di default imposta la proprietà *"Copy Local"* del riferimento a *"False"*, visto che si aspetta di trovare la *dll* nella GAC, ma (come mi è capitato spesso) se la *dll* non viene trovata a runtime, conviene impostarla a *"True"* in modo da averne una copia in locale (come in Fig. 2).
A questo punto siamo pronti ad implementare il servizio.
Innanzitutto bisogna importare i namespace:

```
Imports System.IO
Imports System.Web.Services
Imports Microsoft.Web.Services
```

Segue poi la dichiarazione della classe e la *Region* del codice generato dal designer.

```
<System.Web.Services.WebService(Namespace:="http
          ://tempuri.org/TransferSvc/Transfer")> _
Public Class Transfer
  Inherits System.Web.Services.WebService
" Web Services Designer Generated Code "
```

Per semplificare la gestione dei possibili codici di ritorno si utilizza un *Enum*

```
  Private Enum ReturnCodes As Integer
    TransferSucceeded = 0
    NothingToTransfer = -1
  End Enum
```

A questo punto segue il metodo principale:

```
  <WebMethod(Description:="This method is used to
         Transfer Data over the HTTP Channel")> _
    Public Function TransferData(ByVal
              OriginalMessageID As Int32, ByVal
         OptionalMessage As String) As Integer
    Dim numAttachments As Integer =
  HttpSoapContext.RequestContext.Attachments.Count()
    If numAttachments = 0 AndAlso
                      OptionalMessage = "" Then
      Return ReturnCodes.NothingToTransfer
    End If
    Dim destDirName As String =
              System.Configuration.Configuration
                Settings.AppSettings("Directory")
    destDirName &= "_" & OriginalMessageID & "_"
                        & Guid.NewGuid().ToString
    Directory.CreateDirectory(destDirName)
    If numAttachments > 0 Then
      For Each att As Dime.DimeAttachment In
          HttpSoapContext.RequestContext.Attachments
        SaveAttachedFile(destDirName & "\" &
                 IO.Path.GetFileName(att.Id), att)
      Next
    End If
    If OptionalMessage <> "" Then
      SaveTextFile(destDirName & "\" &
          "OptionalMessage.txt", OptionalMessage)
    End If
```

Properties

Microsoft.Web.Services Reference Properties

(Name)	Microsoft.Web.Services
Copy Local	True
Culture	
Description	Microsoft.Web.Services.dll
Identity	Microsoft.Web.Services
Path	C:\Program Files\Microsoft WSE
Runtime Version	v1.0.3705
Strong Name	True
Type	Assembly
Version	1.0.0.0

***Fig. 2:** Proprietà assembly Microsoft.Web.Services: è bene impostare a true l'opzione per la copia locale.*

GLOSSARIO

WS-ATTACHMENTS E COM

Se si vuole utilizzare un client COM (ad esempio scritto in VB6 o altri) per inviare file al nostro servizio, bisogna usare il *Microsoft SOAP Toolkit 3.0* che supporta DIME. Bisogna però apportare alcune modifiche al file *WSDL* del servizio, modifiche che prevedono l'inserimento degli elementi relativi a DIME che WSE non genera automaticamente e che sono necessari al SOAP Toolkit.

```
        Return ReturnCodes.TransferSucceeded
    End Function
```

Come si vede viene sfruttato l'oggetto *HttpSoapContext* per accedere alla collezione degli attachments. La cartella viene creata a partire dalla posizione indicata nel *web.config*, a cui viene aggiunto l'ID inviato dal client, e un GUID (identificativo univoco a 128bit) per evitare duplicazioni.
Ecco infine i due metodi di supporto per salvare il messaggio di testo opzionale e il singolo attachment ricevuto.

```
    Private Sub SaveTextFile(ByVal FileName As String,
                                   ByVal Content As String)
        Dim outFile As IO.TextWriter
        If File.Exists(FileName) Then
            File.Delete(FileName)
        End If
        outFile = File.CreateText(FileName)
        outFile.Write(Content)
        outFile.Close()
    End Sub
    Private Sub SaveAttachedFile(ByVal FileName As
                  String, ByRef att As Dime.DimeAttachment)
        Dim data As Byte()
        Dim outFile As IO.FileStream
        ReDim data(CInt(att.Stream.Length))
        att.Stream.Read(data, 0, CInt(att.Stream.Length))
        If IO.File.Exists(FileName) Then
            IO.File.Delete(FileName)
        End If
        outFile = IO.File.OpenWrite(FileName)
        outFile.Write(data, 0, data.Length)
        outFile.Close()
    End Sub
End Class
```

Naturalmente l'utente ASP.NET (o l'utente configurato al suo posto) deve avere il permesso di creare file e directory nel path specificato.

CREIAMO UN CLIENT PER INVIARE I FILE

Passiamo ora a creare un semplice client per inviare i file al nostro servizio. Creiamo un progetto di tipo *Windows Forms*, e disegnamo la form di Fig. 3.
Anche in questo caso bisogna configurare il progetto per supportare WSE, bisogna quindi selezionare il progetto, tasto destro, *WSE Settings* e in questo caso selezionare solo il primo check-box della prima pagina.
Il tool aggiornerà con la *Microsoft.Web.Dll*, le Reference (ricordarsi di impostare *Copy Local* a *"True"* se a runtime non viene trovata), e inoltre modificherà il *Wizard Add Web Reference* (Aggiungi riferimento Web) per generare non solo il proxy standard, ma anche un proxy modificato che permetterà di accedere alle proprietà aggiuntive. Facciamo quindi una *Add Web Reference* del nostro servizio precedentemente creato, ricordandoci di rinominare *"localhost"* in *"TransferSvc"*. Se si va a vedere il file *Reference.vb* che viene generato, si vede che oltre alla classe "normale" chiamata *Transfer*, viene creata anche una classe *TransferWSE* identica alla prima, ma che eredita dalla *WebServicesClientProtocol:*

Fig. 3: **La struttura della form.**

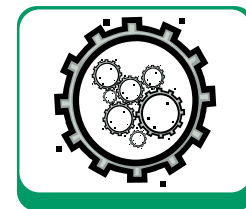
```
    Public Class TransferWse
        Inherits Microsoft.Web.Services.
                                    WebServicesClientProtocol
```

Per utilizzare il WSE lato client dobbiamo creare un proxy appartenente a questa classe.
Prepariamo nel codice della Form una lista che conterrà i nomi dei file da trasmettere:

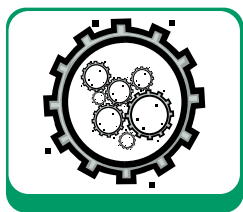
```
Private lista As New ArrayList(0)
```

Siamo pronti a questo punto a mettere il codice per gestire gli eventi:

```
    Private Sub btnAttachFile_Click(ByVal sender
        As System.Object, ByVal e As System.EventArgs)
                                 Handles btnAttachFile.Click
        If OpenFileDialog1.ShowDialog() =
                                      DialogResult.OK Then
            Me.lista.Add(OpenFileDialog1.FileName)
            Me.lblCountAttach.Text = "# of attached files:
                                  " & Me.lista.Count.ToString
        End If
    End Sub
```

Questo metodo serve ad aggiungere il file selezionato alla lista. Seguono i metodi che gestiscono l'invio vero e proprio.

WS-ATTACHMENTS E JAVA

Un'implementazione dello standard DIME per Java è disponibile gratuitamente presso http://onionnetworks.com/developers/

Anche AXIS supporta DIME: http://webservices.apache.org/axis

```
Private Sub btnSend_Click(ByVal sender As
            System.Object, ByVal e As System.EventArgs)
                              Handles btnSend.Click
    Dim TransferSvc As New TransferSvc.TransferWSE
    If lista.Count > 0 Then
        For i As Int32 = 0 To lista.Count - 1
            TransferSvc.RequestSoapContext.
                Attachments.Add(GetFileAsAttachment(
                                lista(i).ToString()))
        Next
    End If
    MessageBox.Show(TransferSvc.TransferData(
                        CInt(Me.txtMessageID.Text),
              Me.txtOptionalMessage.Text).ToString)
End Sub
Private Function GetFileAsAttachment(ByVal FullPath
                    As String) As Dime.DimeAttachment
    Dim stream As IO.FileStream =
                          IO.File.OpenRead(FullPath)
    Dim attach As New
                  Dime.DimeAttachment(FullPath, "0",
          Dime.TypeFormatEnum.MediaType, stream)
    Return attach
End Function
```

Come si vede, attraverso il proxy *TransferSvc.TransferWSE* creato dal WSE Settings Tool, possiamo accedere anche lato client alla collezione degli attachment. Il file viene letto e trasformato in un oggetto di tipo *DimeAttachment*, che viene poi aggiunto alla collezione. L'ID del *DimeAttachment* viene settato al path del file da trasferire, informazione che il servizio estrae per rigenerare il nome del file originale. Lascio a voi l'esercizio di ripulire la form del client e la lista dopo un invio.

DOWNLOAD E ALTRI RIFERIMENTI

Sul sito http://msdn.microsoft.com/webservices sono disponibili sia il WSE 1.0 SP1, sia il *WSE Settings Tools*, ed inoltre ci sono una serie di articoli e alcuni forum su cui approfondire questi argomenti.

IMPOSTIAMO UN ROUTER PER I MESSAGGI

Capita molto spesso che un server venga spostato, rinominato, o modificato, rendendo impossibile ai vecchi client il raggiungimento dei suoi servizi.
Certo, in alcuni casi basta impostare correttamente il DNS e tutto torna a funzionare, ma in altri casi è necessario impostare un nuovo server che funga da router, magari nel caso si sia spostato il vecchio server all'interno di una sottorete che non è più raggiungibile dall'esterno.
Si può in questo caso agire in due modi: preparando a mano un servizio che ha lo stesso nome e gli stessi parametri del precedente e che reinstrada la comunicazione (facendo attenzione a non perdere gli attachment per la strada…), oppure si può sfruttare *WS-Routing*. *WS-Routing* è lo standard che permette di instradare dinamicamente i messaggi a seconda di certe condizioni. Nel nostro caso utilizzeremo *WS-Routing* nella maniera più semplice, reinstradando la chiamata ad una Virtual Directory verso un'altra Virtual Directory, ma il tutto può avvenire tra server diversi, anche più di uno, e può anche avere condizioni che permettono di instradare dinamicamente le richieste verso più server. Per cominciare bisogna creare un progetto Web Service, e poi eliminare il servizio che il Wizard crea, tanto non viene utilizzato.
Si aggiungerà poi al progetto un file chiamato *referralCache.config* (il nome può anche essere diverso) con il seguente contenuto:

```
<?xml version="1.0" ?>
<r:referrals
    xmlns:r="http://schemas.xmlsoap.org/ws/2001/10/
                                          referral">
    <r:ref>
    <r:for>
        <r:exact>http://localhost/TransferRouterSvc/
                                       Transfer.asmx
        </r:exact>
    </r:for>
    <r:if/>
    <r:go>
        <r:via>http://localhost/TransferSvc/Transfer.asmx
        </r:via>
    </r:go>
    <r:refId>uuid:abadf00d-1234-5678-9abc-
                             defgabcdefab</r:refId>
    </r:ref>
</r:referrals>
```

dove la prima URI è quella del nostro router e la seconda URI è quella di destinazione.
Si invoca a questo punto il WSE Settings Tool, si abilita il primo check-box della prima pagina e poi si va sulla pagina Routing. Si seleziona *Add* e si configura il Router come in Fig. 4. Ricordatevi di modificare **.ashx* in **.asmx* nel secondo punto, altrimenti non funzionerà nulla.

Fig. 4: *Configurazione del Routing Handler.*

A questo punto si seleziona *Browse* e lo si fa puntare al file *referralCache.config* creato precedentemente, come in Fig. 5.

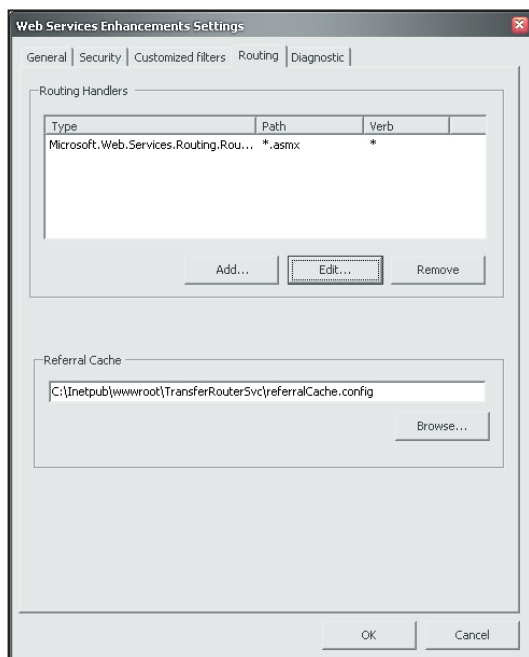



***Fig. 5:* Configurazione finale Routing.**

Il WSE Settings Tool modificherà il web.config abilitando WS-Routing e impostando di prendere i path di instradamento dal file *referralCache.config*. Per testare che il routing funziona si può a questo punto modificare il client indicandogli l'URL del router subito dopo aver creato il proxy:

```
Dim TransferSvc As New TransferSvc.TransferWse
    TransferSvc.Url = "http://localhost/
                TransferRouterSvc/Transfer.asmx"
```

Se il routing è in funzione, per modificare il file *referralCache.config* bisogna o fermare IIS, oppure salvare il file con un altro nome e modificare il *web.config* di conseguenza, in quanto il file *referralCache. config* viene bloccato dal servizio di routing.

E LA SECURITY?

Purtroppo in questo articolo non c'è più spazio per parlare di security, ma volevo ricordare solo due cose: in questo progetto tutte le trasmissioni avvengono in chiaro, a meno di impostare il servizio su HTTPS. Inoltre non viene assolutamente effettuata l'autenticazione del chiamante, permettendo a chiunque di intasare il server con file inutili.
Quindi per il momento conviene utilizzarlo solo su un canale sicuro, in una intranet, su una VPN o un canale HTTPS point-to-point, oppure ci si deve rivolgere alla security di IIS, ricordandosi di modificare il codice del client per passare Username e Password. In un futuro articolo verrà trattato anche lo standard *WS-Security*, permettendo quindi di superare le limitazioni attuali.

MIGLIORIAMO IL PROGETTO

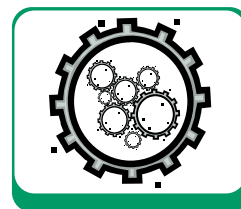
Naturalmente, il progetto presentato in questo articolo è solo la dimostrazione che file binari possono essere inviati tramite Web Services, e può essere notevolmente migliorato. Se provate a inviare file molto grossi, vi scontrerete presto con il limite di 4096Kb come dimensione massima di un invio. Anzi, il possibile invio è anche di qualche KB inferiore perché bisogna tenere conto del documento SOAP… Per ovviare al problema bisogna cambiare le impostazioni di ASP.NET.
Se si guarda nel file *machine.config* si può trovare il seguente elemento:

```
<httpRuntime executionTimeout="90"
 maxRequestLength="4096"
 useFullyQualifiedRedirectUrl="false"
 minFreeThreads="8"
 minLocalRequestFreeThreads="4"
 appRequestQueueLimit="100"
 enableVersionHeader="true" />
```

Bisogna intervenite sull'attributo *maxRequestLength*, che impostato a 4096 indica appunto il limite di 4096Kb. Si può modificarlo a livello del *machine. config*, ma riguarderà l'intera macchina, oppure applicarlo al *web.config* del servizio. In questo modo non si intaccherà il resto delle Web Application.
Un'altra miglioria consiste nel trasformare il client da sincrono ad asincrono, utilizzando i metodi *BeginTransferData* ed *EndTransferData* al posto di *TransferData*.
Naturalmente in questo caso la gestione del client diventa un po' più complicata, ma l'utente può continuare ad usare il programma anche mentre l'invio è in corso.

CONCLUSIONI

Il servizio implementato è solo un primo passo che mostra l'uso di due dei nuovi standard legati al mondo in continua evoluzione dei Web Services: *WS-Attachments* e *WS-Routing*.
La documentazione e i sample forniti con il WSE1.0 SP1, sono veramente completi e ben fatti, e contengono anche tutti i passi da svolgere manualmente nel caso in cui non si voglia utilizzare il wizard offerto dal WSE Settings Tool.

Lorenzo Barbieri

WSE SETTINGS TOOL

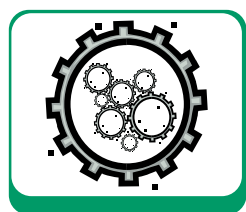
La documentazione del *WSE Settings Tool* è un file HTML presente nella cartella *Unsupported/WSE Settings* dentro la cartella dove viene installato il WSE. Il file non viene linkato da nessuna parte, quindi vi consiglio di aggiungerlo allo *start menu* nella sezione del WSE.

Per qualsiasi delucidazione o informazione vi invito a contattarmi attraverso il mio blog: http://weblogs.asp.net/lbarbieri.

Creazione dell'archivio e aggiunta di file

Un WinZip con Java

parte quinta

Il software SwingZIP sarà completato con l'aggiunta delle funzionalità indispensabili per la creazione di un nuovo archivio e per l'aggiunta di file ad un archivio già esistente.

Siamo finalmente giunti all'ultima parte di questo tutorial, dedicato allo sviluppo di un simil-WinZip multipiattaforma realizzato con Java. Nel corso delle puntate precedenti, oltre ad aver fatto conoscenza del pacchetto *java.util.zip*, abbiamo allestito gran parte dell'applicazione *SwingZIP*. In questo momento il nostro software è completo dal punto di vista della lettura di un file ZIP: può esplorare i contenuti di un archivio ed estrarne totalmente o parzialmente il contenuto. Il mese scorso abbiamo iniziato ad affrontare i problemi legati alla scrittura e alla riscrittura di un archivio ZIP, dotando l'applicazione della possibilità di rimuovere una o più entry da un archivio esistente. Per completare l'opera mancano ancora due fondamentali funzioni: la creazione di un nuovo archivio e l'aggiunta di file ad un archivio già esistente.

CREAZIONE DI UN NUOVO ARCHIVIO

Creare un nuovo archivio inizialmente vuoto non è complicato: è sufficiente far scegliere all'utente il nome e la posizione del file da generare, fare i dovuti controlli di sicurezza e quindi creare semplicemente un archivio ZIP privo di contenuti. C'è un solo problema: per le librerie di Java, un archivio ZIP privo di contenuti non è un archivio valido. L'utente, all'atto della creazione del nuovo file, deve inserire al suo interno almeno un elemento. Non è difficile risolvere il problema, giacché a breve andremo a realizzare una routine utile per la compressione di file e directory all'interno di un archivio. Sarà sufficiente richiamare questa routine prima della creazione del nuovo archivio desiderato dall'utente. Per il momento andiamoci ad occupare del metodo *zipCreate()*:

```
private void zipCreate() {
// Fa scegliere il file all'utente.
JFileChooser fileChooser = new JFileChooser();
fileChooser.setFileFilter(new ZIPFileFilter());
int option = fileChooser.showSaveDialog(this);
// Controlla che l'utente non abbia annullato l'operazione.
if (option != JFileChooser.APPROVE_OPTION) return;
// Acquisisce il file selezionato.
File selectedFile = fileChooser.getSelectedFile();
// Controlla e sistema il nome del file.
if (!selectedFile.getName().toLowerCase().endsWith(
                                                ".zip")) {
   selectedFile = new File(
           selectedFile.getAbsolutePath() + ".zip");}
// Controlla l'esistenza del file.
if (selectedFile.exists()) {
   if (selectedFile.isDirectory()) {
      JOptionPane.showMessageDialog(
         this, "Il file selezionato è una directory",
         "Errore!", JOptionPane.ERROR_MESSAGE);
      return;
   } else {
      int c = JOptionPane.showConfirmDialog(
         this, "Il file selezionato già esiste. Sovrascrivo?",
         "Richiesta conferma",
                              JOptionPane.YES_NO_OPTION);
      if (c != JOptionPane.YES_OPTION) return;}}
// Richiama la routine per l'aggiunta dei file.
addRoutine(false, selectedFile); }
```

Il metodo è estremamente semplice. Per prima cosa si fa ricorso ad un oggetto *JFileChooser*, che l'utente deve utilizzare per scegliere il nome e la posizione del nuovo archivio. Il codice seguente verifica l'informazione fornita dall'utente, accertandosi che l'archivio possa essere effettivamente scritto. Se l'utente dimentica l'estensione *.zip*, il software la aggiunge automaticamente al nome del file. Nel caso in cui l'utente abbia selezionato un archivio già esistente, *SwingZIP* chiederà conferma prima di sovra-

scriverlo. Il metodo *zipCreate()*, ad ogni modo, in nessuna parte del suo codice genera il file desiderato dall'utente. Andiamo ad osservare l'ultima riga di codice:

```
addRoutine(false, selectedFile);
```

Come si è detto poco sopra, non possiamo generare un archivio ZIP completamente vuoto. Per questo motivo *zipCreate()* richiama il metodo *addRoutine()*, fornendogli un paio di parametri. Della realizzazione di questo metodo ci occuperemo immediatamente, e lo progetteremo appositamente per essere utilizzato tanto ora da *zipCreate()* quanto successivamente da *zipAdd()*. Prima di andare a pensare e a realizzare il codice di *addRoutine()*, ad ogni modo, è necessario predisporre alcune classi che ci serviranno tanto per il trasporto delle informazioni quanto per il disegno di una nuova parte dell'interfaccia di *SwingZIP*.

LA CLASSE ADDELEMENT

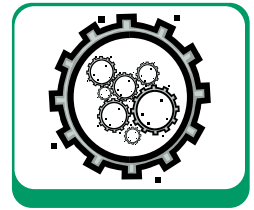
Cominciamo dalla semplicissima classe *AddElement*:

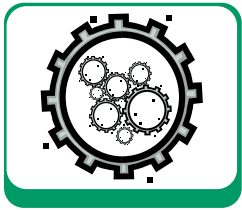
```
import java.io.*;
class AddElement {
  private File file;
  private String path;
  public AddElement(File file, String path) {
    this.file = file;
    this.path = path;}
  public File getFile() {
    return file;}
  public String getPath() {
    return path;}
  public String toString() {
    return path + " [" + file.getAbsolutePath() + "]";}
  public boolean equals(Object o) {
    if (o instanceof AddElement) {
      AddElement element = (AddElement)o;
      return path.toLowerCase().equals(
                              element.path.toLowerCase());
    } else return false; } }
```

La classe *AddElement* è utile per incapsulare all'interno di una sola struttura due informazioni differenti: il riferimento ad un elemento del file system che dovrà successivamente essere compresso in un archivio ZIP ed il percorso che questo avrà internamente all'archivio una volta che sarà stato trasformato in una entry. L'unica parte della classe degna di nota è la ridefinizione del metodo *equals()*, la cui formulazione originaria viene ereditata da *Object* ed il cui scopo è il paragone dell'oggetto di invocazione con un qualsiasi altro oggetto. Secondo la ridefinizione che ne è stata fatta, due oggetti *AddElement* sono considerati equivalenti quando il loro percorso come entry dell'archivio ZIP è lo stesso (in maniera case-insensitive, tra l'altro), anche nel caso in cui i file riferiti dalle apposite proprietà dell'oggetto sono distinti. Questo accorgimento, lo vedremo meglio a breve, ci semplificherà la scrittura del codice che dovrà occuparsi di verificare l'assenza di omonimia tra le nuove entry desiderate dall'utente.

LA CLASSE ADDDIALOG

Ad un certo punto l'utente dovrà scegliere quali sono i file che vuole comprimere all'interno dell'archivio in gestione. Finora non abbiamo realizzato alcuno strumento di interfaccia che consenta tale operazione. Giunge il momento di realizzarlo. La classe *AddDialog* costruisce una finestra di dialogo modale che l'utente potrà impiegare per editare la lista dei file da aggiungere ad un archivio:

```
// Per l'interfaccia grafica.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;
// Per manipolare i file.
import java.io.*;
// Utilità.
import java.util.*;
class
  AddDialog
extends
  JDialog
implements
  ActionListener,
  ListSelectionListener {
    // Elementi di interfaccia.
    JList filesList = new JList();
    JButton button1 = new JButton("Aggiungi");
    JButton button2 = new JButton("Elimina");
    JButton button3 = new JButton("Fatto");
    // Vettore sul quale sono conservati i file scelti
                                              dall'utente.
    Vector files = new Vector();
    // Flag per la conferma dell'operazione.
    boolean confirmed = false;
    // Costruttore.
    public AddDialog(SwingZIP owner, String title) {
      // Chiamata al costruttore base.
      ... }
    // Mostra la finestra di dialogo, bloccando il thread
                                      chiamante fin quando
    // l'utente non completa l'operazione. Restituisce
```

ARCHIVI ZIP

Java fornisce delle librerie che consentono la totale astrazione dal tipo di compressione impiegata dal formato ZIP. Infatti, utilizzando le API del package *java.util.zip*, non vi ritroverete mai ad avere a che fare con i complicati algoritmi che sono alla base della compressione dei file. Le classi offerte dalla libreria di Java fanno tutto da sole. Tuttavia, se vi interessa scoprire come funzioni effettivamente la compressione ZIP, magari per scrivere le vostre API personali, cominciate dando uno sguardo all'indirizzo Web:
http://www.pkware.com/products/enterprise/white_papers/appnote.html

```
la lista dei file
// selezionati dall'utente, oppure null se nessun
elemento è stato
// scelto oppure nel caso di annullamento
dell'operazione.
public Vector showAddDialog() {
  show();
  if (!confirmed || files.size() == 0) return null;
  else return files;}
// Richiamato ogni volta che uno dei bottoni viene
azionato.
public void actionPerformed(ActionEvent e) {
 ...}
// Richiamato ogni volta che cambia la selezione
nella lista dei file.
public void valueChanged(ListSelectionEvent e) {
  // Abilita/disabilita il tasto "Elimina".
  button2.setEnabled(!filesList.isSelectionEmpty());}
  // Richiamato quando l'utente vuole aggiungere
dei file alla lista.
     private void addFilesDialog() {...}
     // Richiamato per aggiungere alla lista i file e
le directory scelte
     // dall'utente. Il metodo è ricorsivo nel caso
l'argomento file sia
     // una directory.
     private void addFiles(File file, String basePath) {
       ...}
     // Questo metodo rimuove dalla lista i file
selezionati dall'utente.
     private void removeFiles() {
       // Elimina gli elementi selezionati.
       Object[] objects = filesList.getSelectedValues();
       for (int i = 0; i < objects.length; i++) {
         files.remove(objects[i]);
       }
       // Aggiorna la lista mostrata nella finestra.
       filesList.setListData(files);}}
```

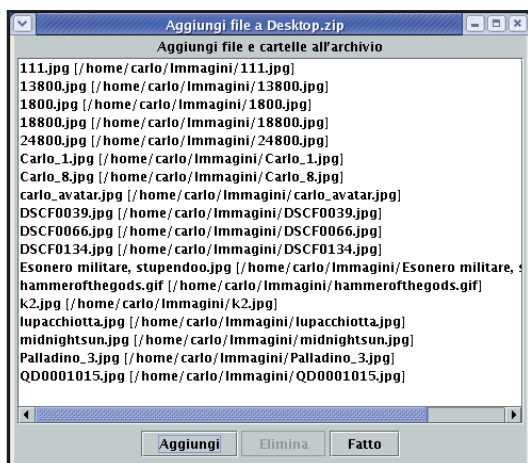



Fig. 1: ***La finestra di dialogo per l'aggiunta di nuovi file ad un archivio.***

Attraverso *AddDialog* l'utente può inserire all'interno di una lista tutti i file e le directory che desidera aggiungere al proprio archivio ZIP. Dal punto di vista del programmatore, la classe è molto semplice. Per richiamarla sarà sufficiente crearne un'istanza e poi appellarsi al metodo *showAddDialog()*. Questo metodo è bloccante, perché bloccante è il metodo *show()* richiamato al suo interno. Stiamo infatti trattando una finestra di dialogo modale. L'utente potrà interagire liberamente con l'interfaccia di *AddDialog* (vedi Fig. 1), ma per tornare al programma principale dovrà prima concludere le operazioni di aggiunta dei file. Quando la finestra di dialogo viene chiusa, o per annullamento o per conferma dell'operazione, *showAddDialog()* ritorna al codice chiamante, restituendogli un oggetto di tipo *java.util.Vector.* Nel caso in cui l'operazione sia stata annullata, oppure se l'utente non ha aggiunto alcun file nella lista, il valore restituito sarà *null*. In tutti gli altri casi il codice chiamate riceverà un *Vector* valido, i cui elementi saranno di tipo *AddElement* ed il cui scopo è la rappresentazione di tutti i file che devono essere aggiunti all'archivio ZIP.

IL METODO ADDROUTINE()

Ora che i due strumenti *AddElement* e *AddDialog* sono stati realizzati, possiamo passare all'analisi di *addRoutine()*, un metodo privato che dobbiamo aggiungere alla principale classe del programma:

```
private void addRoutine(final boolean archiveExists,
                                        final File file) {
 // Richiama la finestra di dialogo per l'aggiunta dei file.
 AddDialog addDialog = new AddDialog(
           this, "Aggiungi file a " + file.getName());
 final Vector addElements = addDialog.showAddDialog();
 // Se nessun elemento deve essere aggiunto, lascia stare.
 if (addElements == null) return;
   // Si rende inattiva l'interfaccia grafica.
   b1.setEnabled(false);
   b2.setEnabled(false);
   b3.setEnabled(false);
   b4.setEnabled(false);
   b5.setEnabled(false);
   // Memorizza un alias della finestra, sostitutivo di
   // this nella classe interna che sta per essere creata.
   final SwingZIP myself = this;
    // Passa all'esecuzione su thread secondario.
    Thread thread = new Thread(new Runnable() {
      public void run() {... }});
 thread.start(); }
```

Cerchiamo anzitutto di comprendere il significato dei due argomenti accettati da *addRoutine():*

- **boolean archiveExists.** Abbiamo già detto che *addRoutine()* sarà richiamato in due differenti casi, cioè per la creazione di un nuovo archivio oppure per l'aggiunta di file ad uno già esistente. L'argomento *archiveExists* distingue i due casi. Se è *true* l'archivio già esiste, e bisogna aggiungere dei file al suo interno; se è *false* l'archivio deve essere generato per la prima volta. Come vedremo a breve, c'è un'importante differenza tra i due distinti casi.

- **File file.** Questo argomento è semplicemente un riferimento all'archivio che dovrà essere creato oppure aggiornato.

Entrambi gli argomenti, come altre variabili definite nella prima parte del metodo, sono *final*. Se avete seguito attentamente le precedenti parti del tutorial già dovreste comprenderne il motivo: le operazioni più pesanti saranno sviluppate in un thread separato, innestato all'interno del metodo sotto forma di classe interna anonima. Per regola, solo le variabili final possono essere lette dall'interno di un thread innestato. Nella prima parte del metodo viene richiamata un'istanza di *AddDialog*, in modo che l'utente possa specificare quali file desidera aggiungere all'archivio ZIP:

```
AddDialog addDialog = new AddDialog(this, "Aggiungi
                                  file a " + file.getName());
final Vector addElements = addDialog.showAddDialog();
if (addElements == null) return;
```

Se nessun file è selezionato dall'utente, il metodo viene abbandonato. In caso contrario si procede alla disattivazione dell'interfaccia, alla memorizzazione di un alias per *this* e al lancio del nuovo thread incaricato delle operazioni più pesanti:

```
b1.setEnabled(false);
b2.setEnabled(false);
b3.setEnabled(false);
b4.setEnabled(false);
b5.setEnabled(false);
final SwingZIP myself = this;
Thread thread = new Thread(new Runnable() {
  public void run() {
    // Codice del thread secondario...}});
thread.start();
```

Andiamo ora a considerare il contenuto del metodo *run()* della classe interna realizzata. Per prima cosa si dichiarano alcuni elementi iniziali:

```
boolean operationDone = false;
ZipFile sourceFile = null;
InputStream in = null;
ZipOutputStream out = null;
```

Segue poi il codice "rischioso", cioè che può sollevare delle eccezioni. Per questo motivo si fa ricorso ad un unico grande blocco di tipo *try ... catch ... finally:*

```
try {
  // Codice rischioso.
} catch (IOException e) {
  // Mostro un messaggio di errore per l'utente.
  showMessage(  "Errore!", "Impossibile portare a
  termine l'operazione", JOptionPane.ERROR_MESSAGE);
} finally {
  // Chiude i canali rimasti aperti.
  if (sourceFile != null) try {
    sourceFile.close();
  } catch (Exception e) {}
  if (in != null) try {
    in.close();
  } catch (Exception e) {}
  if (out != null) try {
    out.close();
  } catch (Exception e) {}
  // Resetta la barra di progresso.
  progressBar.setValue(0);
  // Se l'operazione è compiuta, riapre il file.
  if (operationDone) openRoutine(file);
  else {
    // Si riattivano i pulsanti "Crea" e "Apri".
    b1.setEnabled(true);
    b2.setEnabled(true);
    // Se c'è un archivio aperto riattiva anche gli altri.
    if (archiveExists) {
       b3.setEnabled(true);
       b4.setEnabled(true);
       b5.setEnabled(true); }
  } }
```

Nel caso in cui il codice rischioso (che andremo a vedere tra un istante) generi un'eccezione, il blocco *catch* sarà incaricato di mostrare un messaggio di errore all'utente. Comunque vada, il blocco *finally* compie alcune operazioni conclusive, come la chiusura dei canali eventualmente rimasti aperti. Al termine di tutto, il booleano *operationDone* contiene un valore che informa sul successo dell'operazione. Se tutto è andato a buon termine, si richiama *openRoutine()* per aprire in *SwingZIP* l'archivio creato, per poterne così riscontrare visivamente le modifiche. In caso di fallimento, non resta che riabilitare l'interfaccia grafica. Dentro il blocco *try* è conservato il cuore pulsante del metodo. Prima di esaminarne nel dettaglio il codice, chiariamo alcuni punti fondamentali. Come abbiamo imparato il mese passato, non è possibile alterare realmente i contenuti di un archivio già esistente. Per l'eliminazione di una o più entry, infatti, abbiamo rigenerato l'archivio da capo, escludendo dal suo interno gli elementi non più necessari. Per l'aggiunta di nuove entry, come è facile intuire, il caso è analogo. Ecco meglio spiegato il significato dell'argomento *archiveExists*: in caso di aggiornamento dovremo creare un nuovo archivio, copiare al suo interno le entry estratte dal prece-

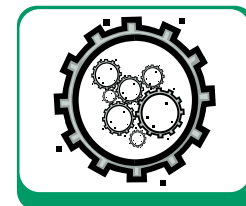
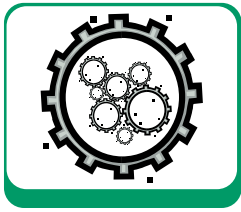



Fig. 2: **Semplicità ed efficacia sono alla base del successo planetario riscosso da WinZip**

Se desideri contattare l'autore di questo articolo scrivi a carlo.pelliccia@ioprogrammo.it

dente, aggiungere i file espressi dall'utente mediante *AddDialog* e, infine, sostituire il vecchio archivio con il nuovo così ottenuto. Nel caso della creazione di un nuovo archivio, invece, tutta la parte relativa alla copia delle entry esistenti non ha più senso e non deve essere considerata. Andiamo a vedere il dettaglio del codice:

```
Vector toCopy = new Vector();
long totalBytes = 0;
long doneBytes = 0;
```

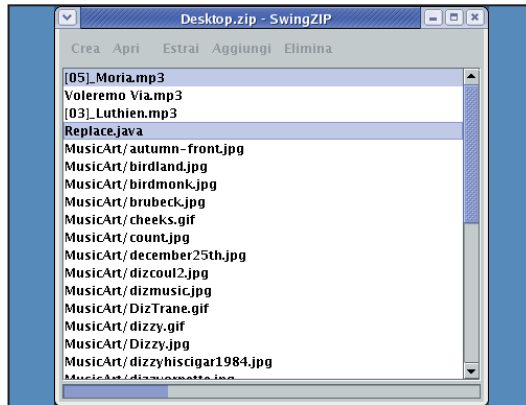



Fig. 3: ***L'interfaccia del nostro JavaZip è completa e ha ben poco da invidiare al prodotto di riferimento.***

Nel vettore *toCopy* saranno conservati i riferimenti alle entry che dovranno essere trasferite da un archivio all'altro. Le variabili *totalBytes* e *doneBytes* serviranno invece per il calcolo del valore da mostrare nella barra di progresso, come già abbiamo fatto in precedenza per casi analoghi.

```
if (archiveExists) {
  sourceFile = new ZipFile(file);
  int c = 0;
  Enumeration entriesEnumeration = sourceFile.entries();
  while (entriesEnumeration.hasMoreElements()) {
    ZipEntry entry = (ZipEntry
                    )entriesEnumeration.nextElement();
    boolean entryExists = false;
    int elementIndex = -1;
    String entryName = entry.getName().toLowerCase();
```

Davanti ad un caso di aggiornamento (*archiveExists* è *true*), l'archivio esistente viene aperto, e tutte le sue entry vengono passate in rassegna.

```
  for (int i = 0; i < addElements.size(); i++) {
    AddElement aux = (AddElement)addElements.get(i);
    if (aux.getPath().toLowerCase().equals(entryName)) {
      elementIndex = i;
      break;} }
```

Ogni entry dell'archivio esistente viene confrontata con i nuovi file che devono essere aggiunti all'archivio. Può infatti capitare che una delle nuove entry abbia lo stesso nome di una già esistente. In questo caso bisogna interpellare l'utente: la nuova entry deve sovrascrivere la vecchia?

```
if (elementIndex > -1) {
  if (c == 0) {
    String[] options = {
      "No (chiedi di nuovo in casi analoghi)",
       "Si (chiedi di nuovo in casi analoghi)",
       "No (non chiedermelo più)",
       "Si (non chiedermelo più)" };
    Object o = JOptionPane.showInputDialog(
           myself, "L'archivio già contiene il file\n\n" +
       entryName + "\n\n" + "Sovrascrivo?","Richiesta
        conferma", JOptionPane.QUESTION_MESSAGE,
                        null, options, options[0] );
  if (o == null || o == options[0] || o == options[2]) {
     toCopy.add(entry);
     totalBytes += entry.getSize();
     addElements.remove(elementIndex);
     if (o == options[2]) c = 1;
  } else if (o == options[3]) c = 2;
} else if (c == 1) {
  toCopy.add(entry);
  totalBytes += entry.getSize();
  addElements.remove(elementIndex);}
} else {
  toCopy.add(entry);
  totalBytes += entry.getSize(); }
  if (addElements.size() == 0) return; }}
```

Ogni entry presente nell'archivio originario viene aggiunta al vettore *toCopy* solo in due casi: quando nessuna delle nuove entry entra in conflitto con essa, oppure quando al verificarsi di un conflitto l'utente boccia la sovrascrittura. In questo secondo caso è la nuova entry ad essere estromessa dal vettore *addElements*. Di ogni entry passata nel vettore *toCopy* viene considerato il peso in byte, aggiungendolo a *totalBytes*. Uno speciale meccanismo di input consente all'utente di non dover essere interrogato troppe volte, e di scegliere una risposta unica per la sovrascrittura di tutte le entry conflittuali. Alla fine del procedimento, un controllo verifica che vi siano ancora elementi all'interno del vettore *addElements*, in modo che l'operazione possa essere interrotta nel caso in cui non siano rimaste entry da aggiungere all'archivio esistente. L'operazione successiva conteggia le dimensioni delle entry da aggiungere all'archivio, aggiornando di conseguenza il valore di *totalBytes*:

```
for (int i = 0; i < addElements.size(); i++) {
  AddElement aux = (AddElement)addElements.get(i);
  totalBytes += aux.getFile().length();}
```

Tutto è stato predisposto. Non resta che passare alla creazione effettiva del nuovo archivio. Si fa uso di un

file temporaneo:

```
File tempFile = File.createTempFile("zip", null,
                              file.getParentFile());
out = new ZipOutputStream(new
                         FileOutputStream(tempFile));
```

Nel caso dell'aggiornamento di un archivio esistente, è il momento di trasferire le entry annotate nel vettore *toCopy* dal vecchio al nuovo archivio:

```
if (archiveExists) {
  for (int i = 0; i < toCopy.size(); i++) {
    ZipEntry entry = (ZipEntry)toCopy.get(i);
    in = sourceFile.getInputStream(entry);
    out.putNextEntry(new ZipEntry(entry.getName()));
    byte[] buffer = new byte[1024];
    int l;
    while ((l = in.read(buffer, 0, buffer.length)) != -1) {
      out.write(buffer, 0, l);
      doneBytes += l;
      progressBar.setValue((int)Math.round(
                (doneBytes * 100D) / totalBytes));}
      out.closeEntry();
      in.close(); }
        sourceFile.close();}
```

Copiate le vecchie entry, bisogna aggiungere le nuove annotate nel vettore *addElements:*

```
for (int i = 0; i < addElements.size(); i++) {
  AddElement addElement =
                   (AddElement)addElements.get(i);
  in = new FileInputStream(addElement.getFile());
  out.putNextEntry(new ZipEntry(
                          addElement.getPath()));
  byte[] buffer = new byte[1024];
  int l;
  while ((l = in.read(buffer, 0, buffer.length)) != -1) {
    out.write(buffer, 0, l);
    doneBytes += l;
    progressBar.setValue((int)Math.round((doneBytes
                     * 100D) / totalBytes)); }
  out.closeEntry();
  in.close(); }
```

Le operazioni di scrittura, a questo punto, sono state completamente effettuate. Il canale verso il nuovo archivio può essere chiuso:

```
out.close();
```

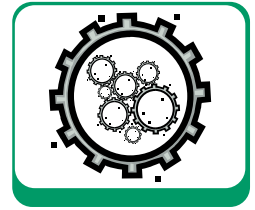
Non resta che spostare il file temporaneo alla posizione cui si aspetta di trovarlo l'utente, rimpiazzando l'eventuale vecchio archivio presente:

```
if (!file.delete() && archiveExists) throw new
                               IOException();
```

```
if (!tempFile.renameTo(file)) throw new IOException();
```

Tutto è stato compiuto. Se l'esecuzione è arrivata fino a questo punto, il successo è garantito. Bisogna annotarlo nell'apposito flag:

```
operationDone = true;
```

Il blocco *try* è completo. Come abbiamo già visto, sarà il blocco *finally* ad eseguire le operazioni finali, per la restituzione dell'uso dell'interfaccia all'utente.

AGGIUNTA DI FILE AD UN ARCHIVIO ESISTENTE

Dopo aver realizzato *addRoutine()* lo sviluppo del codice di *zipAdd()*, il metodo richiamato quando l'utente decide di aggiungere nuovi elementi all'archivio gestito da SwingZIP, diventa davvero banale:

```
private void zipAdd() {
         addRoutine(true, file);}
```

CONCLUSIONI

Con *zipCreate()* e *zipAdd()* il nostro SwingZIP è finalmente completo. Per certi aspetti il software è ancora rudimentale, poiché potrebbe essere dotato di un migliore aspetto grafico e di ulteriori funzionalità. Ad esempio è una buona idea rimpiazzare la *JList* al centro del programma con una *JTable* capace di mostrare i dettagli di ogni entry. A livello funzionale, invece, si potrebbe estendere il lavoro svolto permettendo all'utente la scelta del livello di compressione dell'archivio. Consultando la documentazione ufficiale delle API di Java scoprirete come fare. Ad ogni modo, già così il programma è perfettamente utilizzabile. Lo testimonia questo stesso articolo: dopo averlo riletto lo comprimerò insieme a tutti i file collegati in un solo archivio ZIP da consegnare alla redazione, e lo farò con il programma appena realizzato. Se mi state leggendo vuol dire che SwingZIP funziona! :-)

Carlo Pelliccia

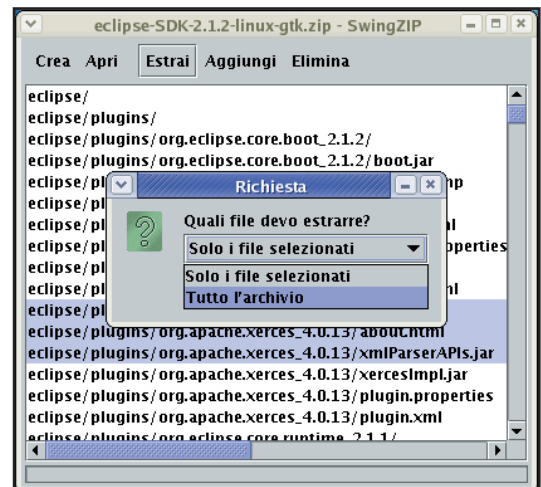
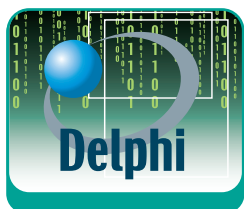



Fig. 4: Le funzionalità del nostro progetto sono volte a fornire la maggiore flessibilità possibile agli utenti.

Realizziamo un'applicazione per indicizzare le parole di un testo

Delphi: corso di Object Pascal

parte quinta

Questo mese è la volta di abbandonare la programmazione lineare e cominciare a studiare tutte quelle estensioni Delphi che hanno trasformato il Pascal in "Object Pascal".

Fino alla metà degli anni 80 lo sviluppo software era interamente dominato dalla programmazione lineare. Questa implicava che dall'inizio della sua esecuzione il codice seguiva un percorso ben stabilito, rispettando il flusso esecutivo dalla prima istruzione fino all'ultima. L'unica cosa che poteva modificare il suo corso erano gli statement condizionali e quelli iterativi, oppure le chiama te alle subroutine (procedure e funzioni), ma era comunque sempre possibile avere sotto controllo il percorso esecutivo che il nostro programma avrebbe seguito. Era uno stile di stesura di codice che si adattava senza problemi anche alle interfacce utente delle prime applicazioni software, in cui la persona seguiva una serie di schermate per lo più testuali in maniera rigidamente guidata e dove l'interazione era fortemente gestita dal programma in esecuzione.

UN MONDO FATTO DI OGGETTI

Fu principalmente con l'avvento delle interfacce grafiche che cambiarono un po' le cose, e soprattutto cambiò la prospettiva con cui si guardava alla scrittura del codice: innanzitutto, tra pulsanti, finestre, mouse che si sposta sullo schermo, e tutto il resto che già conosciamo, non è più il programma a stabilire quale strada l'utente dovrà seguire, ma è l'utente stesso a decidere cosa cliccare, che programma attivare, quale casella di testo riempire, quale pulsante premere, attivando così parti della nostra logica applicativa a seconda del suo bisogno. Inoltre, il crescente utilizzo di soluzioni informatiche per risolvere problematiche sempre più vicine alla quotidianità di persone e piccole imprese rendeva meno adeguata la programmazione lineare e favorì il nascere di una nuova filosofia di lavoro tra i professionisti del settore: si trattava di un framework di sviluppo che voleva modellare le applicazioni sulla base della realtà cui esse tentavano di venire incontro. Si creò così il concetto informatico di *oggetto*, cioè un'entità software che incapsula al suo interno oltre alle procedure per compiere la propria elaborazione, anche i dati su cui deve lavorare. Si abbandonava così l'idea delle applicazioni il cui codice scorreva regolare dall'inizio alla fine per passare ad un concetto più modulare di software, dove vari elementi – ognuno con una funzionalità ben specifica e specializzata – interagivano tra loro nei modi più svariati e, soprattutto, non predefiniti: nasceva così la programmazione orientata agli oggetti! Vi devo confessare che vent'anni fa l'OOP mi aveva spaventato un po' per la sua (apparente) complessità e l'avevo lasciata vergognosamente da parte (la ritenevo molto accademica e complicata per le cose che dovevo fare): è stato solo qualche anno dopo, quando ho dovuto lavorarci per amore o per forza, che ho imparato ad apprenderne le sfumature e le potenzialità, ed oggi mi tocca ammettere che non se ne può fare proprio a meno: praticamente tutto viene realizzato con oggetti, e quei linguaggi che non li gestivano sono stati estesi per poterne fare uso (il C è diventato C++, il Pascal è diventato Object Pascal, il Visual Basic ha introdotto le classi, e perfino il Perl ed il PHP si sono adeguati). Nel resto di questo articolo, quindi, rivedremo i concetti fondamentali dell'OOP e impareremo come si applicano al linguaggio di Delphi.

CLASSI, METODI E PROPRIETÀ

Alla base della programmazione ad oggetti c'è, ovviamente, l'oggetto. Ma per crearne uno abbiamo prima di tutto la necessità di dargli una definizione, di specificare cioè quali sono i dati che lo compongono e quali operazioni esso sarà in grado di com-

piere: questo viene fatto ad opera della classe, che è lo stampino sulla base di cui vengono creati tutti gli oggetti di un certo tipo. Per spiegare questo concetto, possiamo correre indietro fino alle disquisizioni della filosofia greca, da Socrate ad Aristotele e Platone, quando si cercava quel qualcosa di astratto che pur non coincidendo con un qualcosa di concreto ci permette di definirlo: quando parlo di "cavallo" in generale ho in mente un'idea che non corrisponde a nessun cavallo in particolare ma che li comprende tutti. La classe è un po' la stessa cosa. Di per sé non può essere usata per elaborare nulla, ma grazie ad essa possiamo creare quanti oggetti vogliamo ed è su di essi che possiamo poi effettivamente operare: definisce dati ed operazioni, e grazie all'istanziazione viene allocato lo spazio per tali dati e ci viene data la possibilità di eseguire le operazioni – viene creato, insomma, un oggetto (Fig. 1)! È molto importante avere chiara in mente la distinzione tra una classe ed un oggetto, perché l'utilizzo dei due elementi è significativamente differente. Una classe non ha delle risorse fisiche associate sul sistema e non occupa pertanto spazio applicativo, ma definisce quella che sarà la struttura degli oggetti che da essa derivano. Con l'istanziazione di una classe, quella definizione prende forma: tutte le variabili membro vengono allocate, i metodi divengono invocabili e conseguentemente le risorse del sistema occupate per fare spazio alla struttura di dati e metodi che la classe definiva. Da tale definizione possono ovviamente essere creati tutti gli oggetti che vogliamo, i quali risulteranno con lo stesso scheletro strutturale, ma ognuno conterrà una sua copia distinta e separata di dati e metodi.

CREIAMO LE CLASSI

Vediamo dunque di creare due semplici classi, una rappresenta un punto cartesiano, l'altra invece una linea. Entrambe comprendono i dati necessari alla definizione dell'oggetto corrispondente (una coordinata *x* ed una *y* per il punto, due punti per la linea, ed un colore) più un metodo per disegnare tale oggetto sul canvas di un form. Analizziamo insieme il *Listato 1* che contiene il codice di dichiarazione delle due classi.

Listato 1: due semplici classi: un punto e una linea geometrici

```
interface
uses
  Graphics;
type
  TPunto = class
    private
      _x: Integer;
      _y: Integer;
```

```
      function GetCoordX(): Integer;
      procedure SetCoordX(value: Integer);
      function GetCoordY(): Integer;
      procedure SetCoordY(value: Integer);
    public
      // proprietà colore
      colore: TColor;
      // il costruttore della classe
      constructor Create(x: Integer; y: Integer);
      // proprietà coordinata x
      property x: Integer
        read GetCoordX
        write SetCoordX;
      // proprietà coordinata y
      property y: Integer
        read GetCoordY
        write SetCoordY;
      // metodo per disegnare il punto
      procedure Disegna(gdc: TCanvas);
  end;
  TLinea = class
    public
      // proprietà colore
      colore: TColor;
      // proprietà punto iniziale
      punto1: TPunto;
      // proprietà punto finale
      punto2: TPunto;
      // costruttore
      constructor Create(inizio: TPunto; fine: TPunto);
      // metodo per disegnare la linea
      procedure Disegna(gdc: TCanvas);
  end;
```

Notiamo subito che le classi si dichiarano nella sezione *type* di una unità o programma e che la loro sintassi è simile a quella dei *record*, ma si usa la parola chiave *class*. All'interno del blocco *class* possiamo dichiarare tutte le varie proprietà e metodi del tipo di oggetto che stiamo definendo: i cosiddetti membri della classe (appunto, i *metodi* e le *proprietà*) vengono classificati in base alla loro visibilità: abbiamo così i membri *public* (visibili a qualunque codice che istanzi la classe) e quelli *private* (visibili solo all'interno della classe stessa o comunque nel modulo – inteso come file – che la dichiara). Avrete notato che le proprietà (dati) della classe possono essere dichiarate in due modi

Fig. 1: *Una schematizzazione dei concetti di classe, oggetto e istanziazione.*

L'OGGETTO TCANVAS

Ogni form in Delphi mette a disposizione attraverso la proprietà *Canvas* questo oggetto con cui è possibile disegnare linee, rettangoli e altre forme geometriche sulla finestra. Il canvas possiede una penna (*TPen* nella proprietà *Pen*) che è possibile modificare per utilizzare un colore o uno spessore di linee differenti nel disegno. Notate che questo oggetto va utilizzato durante l'evento OnPaint di un form, perché altrimenti quello che si traccia sparisce appena la finestra viene nascosta e ridisegnata (come nel caso della linea creata con il doppio click sulla nostra applicazione).

diversi: come semplici variabili membro oppure attraverso la parola chiave *property.* La differenza fondamentale tra le due modalità è che con le property si definiscono delle routine di recupero ed impostazione del dato, permettendo così di effettuare dei controlli sui valori passati, di calcolare eventualmente il valore restituito, o ancora di avere proprietà di sola lettura o sola scrittura. Nella nostra classe *TPunto,* per esempio, colore è una variabile membro pubblica che può quindi essere letta od impostata direttamente da chi usa un oggetto di quel tipo (il programmatore utente, insomma, può alterare il valore della variabile a suo piacere senza nessun tipo di controllo da parte dell'oggetto), mentre le coordinate *x* e *y* sono memorizzate in due variabili private invisibili al di fuori dell'istanza (*_x* e *_y*) che non sono quindi gestibili esternamente, ma vengono invece mantenute ad opera dei metodi *GetCoordX, GetCoordY, SetCoordX* e *SetCoordY,* dove viene effettuato un controllo per assicurarsi che i valori siano sempre compresi tra 0 e 5000.
La sintassi per una proprietà è la seguente

```
property <nome>:<tipo> read <metodolettura> write
                                     <metodoscrittura>;
```

che deve essere preceduta dalle dichiarazioni dei due metodi indicati, con questi parametri

```
function <metodolettura>:<tipo>;
procedure <metodoscrittura>(<nomevar>:<tipo>);
```

e mentre questi due metodi di accesso possono tranquillamente essere privati e quindi richiamabili solo dall'interno della classe, l'utente finale dell'oggetto potrà leggere o settare la proprietà con una sintassi del tipo

```
ValoreAttuale:= MyObject.MiaProprietà;
MyObject.MiaProprietà:= NuovoValore;
```

dove la cosa importante da notare è il punto (.) qualificatore, da usare sempre per associare un membro di una classe all'oggetto da cui lo stiamo richiamando (come per i record). Omettendo la clausola *read* avremo una proprietà *write-only,* mentre omettendo l'altra ne otterremo una *read-only* (caso decisamente più comune). Vi sarete anche accorti che all'interno della definizione di una classe noi mettiamo solo le dichiarazioni degli eventuali metodi che ne fanno parte: la loro implementazione dovrà seguire successivamente fuori dalla classe, ma necessariamente nello stesso modulo (per esempio, nella sezione *implementation* di una *unit*). Si tratta di una struttura sintattica simile a quella delle routine *forward,* però in questo caso per poter determinare di quale classe stiamo implementando un metodo (nel nostro codice, ad esempio, abbiamo *Disegna* sia per *TPunto* che per *TLinea*), dovremo utilizzare la forma qualificata del nome del metodo, quella cioè preceduta dal nome della classe ed un punto:

```
procedure <classe>.<metodo>(…);
```

Per un riscontro pratico, date un'occhiata al codice del progetto allegato alla rivista, dove troverete le due classi del *Listato 1* con implementazione ed un semplice form che le utilizza. Lì troverete anche – negli eventi *OnPaint* e *OnDblClick* – il modo in cui tali classi possono essere utilizzate in *Object Pascal* per creare degli oggetti da manipolare nelle nostre applicazioni, ed è proprio di questo che ci occupiamo.

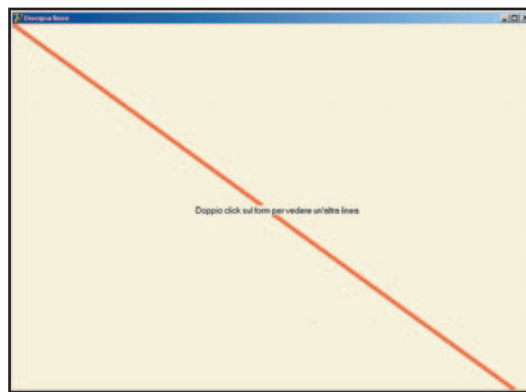



Fig. 2: ***Una banale applicazione che disegna due punti e una linea: ma punto e linea sono degli oggetti! (NB i punti sono praticamente invisibili sul form).***

LA VISIBILITÀ PUBLISHED

Oltre a *private* e *public,* esistono altri ambiti di visibilità dei membri di una classe. Di quello *protected,* che tocca il concetto di ereditarietà, parleremo nel prossimo articolo, mentre gli altri sono fuori dall'obiettivo che ci si è posti per questa serie. Vale la pena di menzionare però la visibilità *published,* che è simile a quella *public,* ma che espone le proprietà ed i metodi della classe per la consultazione a run-time tramite RTTI (a cui avevo già accennato nella serie di due articoli su Web Services e Delphi).

INSTANZIARE LE CLASSI

Una volta definita una classe come abbiamo appena visto, dobbiamo avere degli oggetti prima di poterne utilizzare le funzionalità. Si è già detto che una classe è come un concetto astratto che definisce l'idea del nostro oggetto, ma che per avere un oggetto vero e proprio dobbiamo innanzitutto creare lo stesso attraverso un'istanziazione. Questa avviene invocando dei metodi speciali delle classi detti costruttori, di cui avete due esempi nel *Listato 1.* Questi metodi particolari servono per indicare a Delphi che deve allocare lo spazio per un oggetto così come definito dalla classe su cui è invocato (in effetti il costruttore lo si richiama dal nome della classe stesso, non da una variabile) e ci restituisce un riferimento all'oggetto appena allocato che noi possiamo assegnare ad una variabile del nostro codice per poterne poi richiamare i vari metodi e proprietà. Nell'evento *OnPaint* dell'applicazione allegata trovate il seguente frammento di blocco di istruzioni

```
var
  puntoA: TPunto;
begin
  puntoA:= TPunto.Create(500,500);
```

```
  puntoA.colore:= $00FF00;
  puntoA.Disegna(Canvas);
end;
```

dove potete osservare quanto detto sopra. È possibile definire tutti i costruttori che volete per la vostra classe, al fine di mettere a disposizione di chi userà il vostro codice diverse possibilità di configurazione iniziale dell'oggetto che crea. In questo caso, è sufficiente posporre la clausola *overload* a tutti i costruttori definiti e la stessa cosa la si può fare con qualunque altro metodo, poiché il concetto di overloading (che abbiamo visto nella terza parte di questo corso) si applica anche alle routine interne delle classi. Per *TLinea,* ad esempio, potremmo offrire le seguenti alternative, per poter creare un punto: senza parametri, oppure passando *x* e *y,* oppure ancora passando *x, y* e *colore,* e per disegnarlo con una linea di spessore indicato o di default (il *TCanvas* di *Disegna* viene invece recuperato dal form ed è necessario per poterci disegnare sopra):

```
constructor Create(); overload;
constructor Create(inizio: TPunto; fine: TPunto); overload;
constructor Create(inizio: TPunto; fine: TPunto; col:
                                          TColor); overload;
procedure Disegna(gdc: TCanvas); overload;
procedure Disegna(gdc: TCanvas, width: Integer);
                                                  overload;
```

Notate, comunque, che sarebbe un grave errore utilizzare la variabile *puntoA* senza averla precedentemente inizializzata con una chiamata al costruttore della classe.

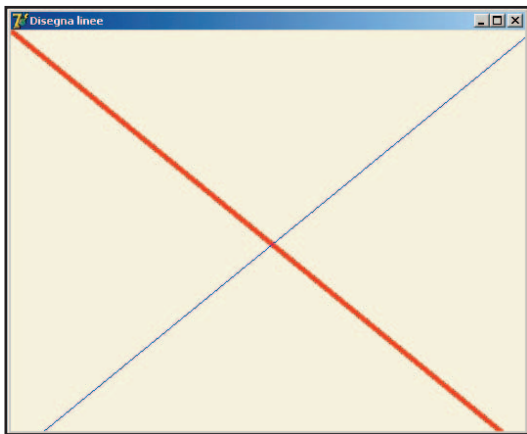



Fig. 3: ***Doppio click sul form e viene instanziato un altro oggetto TLinea, ma con proprietà differenti…***

INCAPSULAMENTO: IL CONCETTO OOP DEL GIORNO

Vediamo di definire insieme, un po' alla volta, i capisaldi della programmazione ad oggetti. Dopo aver visto rapidamente come si crea una classe e come la si utilizza sotto forma di oggetto nel proprio codice, occupiamoci dell'importanza di programmare secondo la logica orientata agli oggetti. L'utilizzo di questa metodologia ci permette di modellare le nostre applicazioni in maniera più simile alla realtà che ci circonda. Ma in particolare ci fa lavorare secondo degli schemi modulari, con blocchi di codice indipendenti e riutilizzabili come componenti a sé stanti che interagiscono tra loro. Senza chiamarlo per nome, abbiamo già visto in opera un primo concetto fondamentale dell'OOP, quello di incapsulamento: con questo termine si intende la caratteristica per cui gli oggetti contengono al loro interno sia i dati che le operazioni sui dati stessi, divenendo così entità indipendenti dal tutto e a sé stanti. Non solo! Avete visto che con la visibilità privata è anche possibile nascondere determinati dettagli di implementazione a chi utilizza l'oggetto, lasciando così aperta la possibilità di migliorare o cambiare completamente il codice della nostra classe, senza dover alterare le applicazioni che la utilizzano, a patto che i nostri metodi e proprietà pubblici restino invariati. Oltre all'incapsulazione, che era il tema di questo mese, tutti i linguaggi orientati agli oggetti ci offrono caratteristiche quali ereditarietà, polimorfismo, astrazione, che rendono questo stile di programmazione molto produttivo. Nei prossimi numeri ci soffermeremo sulla possibilità di creare delle intere gerarchie di classi e vedremo così come estendere la nostra unità di figure geometriche per fornire tutta una serie di oggetti con i quali illustreremo molte altre potenzialità delle tecniche OOP.

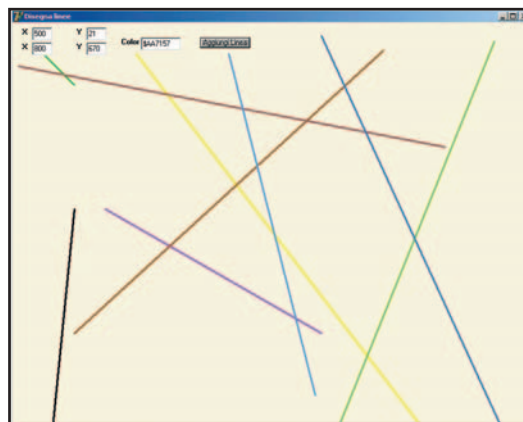
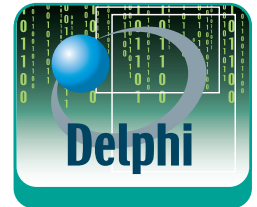



Fig. 4: ***Una sofisticazione sull'applicazione di base, ma sempre usando gli stessi oggetti: create tutte le linee che volete.***

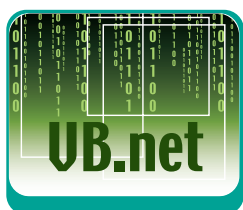
Vi aspetto quindi al mese prossimo per continuare la nostra avventura, ma nel frattempo potete studiare l'applicazione aggiuntiva che ho allegato a questo numero (Fig. 4), dove insieme agli oggetti mettiamo anche gli array e creiamo al volo tutte le linee che vogliamo.

Federico Mestrone

DICHIARAZIONE DELLE CLASSI

Notate che nel dichiarare una classe ed i suoi membri, le variabili semplici devono sempre precedere tutte le proprietà dichiarate con property e tutti i metodi, in ognuna delle sezioni di visibilità (private, public, etc). In caso contrario Delphi rifiuterà di compilare la classe con un errore di compilazione.

Il modello ADO.NET e le tecniche di accesso ai dati

I componenti per l'accesso ai dati

Utilizzando un semplice database Access, impareremo a collegare un controllo *DataGrid* ad una sorgente dati e a visualizzare i dati in maniera corretta, consentendo l'interazione degli utenti.

Collegare un controllo ad una sorgente dati senza scrivere codice specifico, significa eseguire il binding dei dati. Il controllo riceverà le sue informazioni dai componenti di accesso ai dati, in modo che possano essere mostrati automaticamente senza scrivere una riga di codice ed eventualmente manipolati. In questo articolo vedremo come visualizzare i dati di un Dataset in un controllo *DataGrid* utilizzando ancora una volta il database Microsoft Access: *GestioneContatti*. Descriviamo brevemente i controlli disponibili nella scheda *Dati* della casella degli strumenti.

STABILIRE LE CONNESSIONI CON OLEDBCONNECTION

Il componente *OleDbConnection* consente di stabilire una connessione ad un'origine dati specifica, permettendo di aprire e chiudere una connessione ad un database. Per il suo funzionamento si devono specificare alcune informazioni quali il nome e la posizione del database ed il driver da utilizzare per connettersi al database. Per inserire un componente *OleDbConnection* in un Form, è sufficiente trascinarlo su di esso ed impostarne le proprietà. Più semplicemente, si può aggiungere un componente *Ole-DbDataAdapter* al form, in modo da visualizzare una procedura guidata che aggiungerà automaticamente *Ole-DbConnection* con le proprietà impostate durante la procedura, come vedremo in seguito.

NOTA

Tutte le procedure guidate di Visual Basic .NET iniziano con una schermata di benvenuto, che descrive in sintesi le azioni che verranno eseguite dalla procedura guidata e le informazioni che verranno richieste. E' possibile proseguire con la procedura guidata, oppure annullarla, cliccando sui pulsanti corrispondenti.

LA COMUNICAZIONE DEI DATI CON OLEDBDATAADAPTER

Il componente *OleDbDataAdapter* consente la comunicazione tra una fonte dati ed un oggetto *DataSet* e risolve gli aggiornamenti con l'origine dati. Questo componente viene quindi utilizzato non solo per recuperare i dati dal database, ma anche per inserire, aggiornare ed eliminare dati nel database. Il componente *OleDbDataAdapter* verrà utilizzato semplicemente per recuperare i dati da un database e popolare il componente *DataSet*.

IL COMPONENTE DATASET

Il componente *DataSet* rappresenta l'oggetto principale dell'architettura disconnessa di *ADO.NET.*
Il *DataSet* è paragonabile ad un database relazionale di piccole dimensioni conservato in memoria sul client, e non dipende da alcun database specifico. Poiché il componente *DataSet* conserva tutti i dati in memoria, è possibile navigare nei dati sia in avanti sia all'indietro ed operare anche degli aggiornamenti. Con il metodo *Fill* si popola il *DataSet,* mentre con il metodo *Update* si aggiorna il database con i record modificati nel *DataSet.* In questo articolo utilizzeremo questi primi tre componenti appena descritti.

OLCDBCOMMAND: PER IMPARTIRE ISTRUZIONI SQL AL DB

Il componente *OleDbCommand* permette di eseguire istruzioni SQL sul database. È possibile eseguire query d'interrogazione oppure inviare un qualsiasi comando sql come un'istruzione *insert, update* o *delete*.

COMPONENTI SPECIFICI PER SQL SERVER

I componenti *OleDbConnection, OleDbCommand*

ed *OleDbDataAdapter* sono progettati per essere utilizzati su qualsiasi database che supporta *OLE DB,* inoltre dispongono di componenti corrispondenti all'uso con Microsoft SQL Server, ossia *SqlConnection, SqlCommand* ed *SqlDataAdapter.* Questi componenti sono progettati per essere utilizzati esclusivamente con Microsoft SQL Server ed offrono prestazioni migliori rispetto alle controparti *OleDb,* che sono più generiche.

UNA VISTA PERSONALIZZATA

Il componente *Dataview* rappresenta una vista personalizzata dei dati in un componente DataSet. È possibile creare una visualizzazione di dati specifici contenuti in un componente DataSet oppure filtrare, ordinare, modificare o spostare i dati di una tabella senza influenzarne i valori.

IL CONTROLLO DATAGRID

Il controllo *DataGrid* viene utilizzato per visualizzare i dati in una griglia scorrevole in maniera molto simile a quando visualizzate i risultati di una query in Access. È sufficiente associare un componente di accesso ai dati, affinché il controllo visualizzi tutti i dati, permettendone anche la modifica impostando le proprietà opportune.
Tra le proprietà segnaliamo:

- La proprietà *CurrentCell* che permette di determinare la cella selezionata. Restituisce la cella attiva identificata dall'oggetto *DataGridCell.* L'oggetto *DataGridCell* permette di accedere ad una cella nella griglia restituendo il numero di colonna *(ColumnNumber)* e di riga *(RowNumber)* della cella selezionata.
- La proprietà *Item* permette di modificare il valore di una cella. Per referenziare la singola cella è possibile utilizzare indici di riga e di colonna della cella oppure un singolo oggetto *DataGridCell.*
- Le proprietà *CaptionText, CaptionForeColor, CaptionBackColor, CaptionFont* per impostare l'aspetto del titolo (didascalia) del controllo. *CaptionText* imposta il testo da visualizzare nell'area del titolo della griglia. *CaptionForeColor* ne imposta il colore di primo piano. *CaptionBackColor* ne imposta il colore di sfondo. *CaptionFont* ne imposta il tipo di carattere. Infine *CaptionVisible* pari a *True* permette la visualizzazione dell'area, mentre pari a *False* la rende invisibile.
- La proprietà *AlternatingBackColor* permette di impostare il colore di sfondo a righe alternate, questo facilita la visualizzazione dei dati, soprattutto in presenza di molte righe.

Come vedremo, nel momento del suo utilizzo, il *DataGrid* offre l'opportunità di espandere la larghezza delle colonne in fase di esecuzione e di ordinare i dati semplicemente facendo clic sull'intestazione di colonna. Si possono ordinare i dati in ordine crescente o decrescente, ed una piccola freccia rivolta verso l'alto o verso il basso, posta a destra della colonna, indica il tipo di ordinamento. Le proprietà principali che devono essere impostate per associare il componente DataSet al controllo, sono le proprietà *DataSource* e *DataMember.* La proprietà *DataSource* deve essere impiegata per impostare l'origine dei dati che verrà utilizzata per fornire i dati da visualizzare al controllo *DataGrid.* L'origine dei dati può essere uno dei vari componenti di accesso ai dati quali *DataSet, DataView* o *DataTable.* In questo articolo utilizzeremo un componente DataSet. La proprietà *DataMember* permette di selezionare la tabella di un *DataSource* che deve essere utilizzata per visualizzare i dati. Se un oggetto *DataSource* contiene più elementi che forniscono dati è necessario indicare al controllo *DataGrid* quale utilizzare. Se, ad esempio, il *DataSource* è un controllo DataSet che contiene i risultati di due query che elencano i dati delle tabelle contatti ed aziende, è necessario selezionare una delle due tabelle per visualizzarne i dati nel controllo. Potreste visualizzare contemporaneamente i dati di entrambe le tabelle in una relazione genitore-figlio, in questo caso viene visualizzato un segno più *(+)* all'interno del *Datagrid.* Cliccando sul segno più *(+),* viene visualizzato un nodo che contiene i collegamenti alle tabelle figlio.

Fig. 1: ***Configurazione della connessione al database Access.***

CREARE IL PROGETTO GESTIONECONTATTI

Per sviluppare l'applicazione, creiamo un nuovo progetto dal nome *GestioneContatti.* In form1 inseriamo un controllo *Datagrid,* che si trova nella scheda *Windows Form,* ed un controllo *OleDbDataAdapter,* che si trova nella scheda data. Nel momento in cui trasciniamo il controllo *OleDbDataAdapter* sul form, si aprirà la finestra di dialogo Configurazione guidata adattatore dati, in cui si deve cliccare su avanti per proseguire. Il passo successivo della procedura guidata chiede di specificare la connessione al database, permettendo di scegliere una connes-

NOTA

Se non riuscite a connettervi al database, accertatevi di avere selezionato Microsoft Jet 4.0 OLE DB Provider nella scheda Provider e di avere selezionato il database corretto nella scheda Connessione.

sione ai dati esistente, se presente, o di crearne una nuova. La connessione ai dati viene creata utilizzando un oggetto *OleDbConnection* in cui, per connettersi al database, è necessario indicare alcune informazioni come il percorso completo del data-base ed il provider *OLE DB*. Per creare una nuova connessione al database *GestioneContatti.mdb* dobbiamo cliccare sul pulsante *Nuova Connessione* in modo da visualizzare la finestra di dialogo: *Proprietà di Data Link*. La finestra di dialogo *Proprietà di Data Link*, si apre sulla scheda *Connessione*, con le impostazioni da configurare per accedere ad un database SQLServer, per i nostri scopi dobbiamo cliccare sulla scheda *Provider* da cui selezionare la voce: *Microsoft Jet 4.0 OLE DB Provider.* Negli articoli precedenti abbiamo già descritto come i provider offrano, nel loro insieme, un accesso veloce ed affidabile a specifiche origini dati. In particolare il *Microsoft Jet 4.0 OLE DB Provider* permette la connessione ad un database Microsoft Access 2000, mentre potete osservare la lista degli altri provider disponibili che permettono di connettersi a varie origini dati, quali SQL Server o Oracle. A questo punto cliccate sul pulsante *Avanti* alla base della finestra di dialogo per procedere al passo successivo, oppure cliccate sulla scheda *Connessione*. La scheda *Connessione* permette di sfogliare l'albero delle directory e di selezionare il database Access desiderato. Cliccate sul pulsante con i tre puntini posto di fianco alla casella di testo: Selezionare o specificare un nome di database, e selezionate il database *GestioneContatti.mdb* dopo averlo trovato nella solita directory *"c:\mieidatabase"*. Per ciò che riguarda la gestione della sicurezza possiamo accettare le informazioni predefinite. Infine, possiamo cliccare sul pulsante *Verifica Connessione* per avere il riscontro sulla corretta connessione, ed una volta passato il test comparirà la finestra di messaggio relativo al risultato del test. Ora possiamo cliccare sul pulsante *OK* e poi sul pulsante avanti per procedere nella procedura guidata. Nella successiva schermata si deve specificare come si vuole interrogare il database relativamente ai dati. Poiché stiamo utilizzando un database Microsoft Access, sono disponibili solo due opzioni: *Usa istruzioni Sql* ed *Usa stored procedure esistenti*. L'opzione *Usa istruzioni Sql* (default) permette di scrivere un'istruzione *Select*, ed in base alle tabelle incluse nell'istruzione, offre l'opportunità di creare, anche le istruzioni *Update, Insert* e *Delete* corrispondenti. Le istruzioni di aggiornamento del database comprendono i parametri per passare nuovi valori, ed, all'interno di una clausola *WHERE*, anche i parametri per individuare il record corretto nel database. L'opzione *Usa stored procedure esistenti* permette di selezionare una query esistente nel database di Access per ciascuno dei quattro comandi. Naturalmente questa opzione è utile se sono già disponibili query per l'esecuzione delle quattro operazioni su database. Per i nostri scopi lasciamo l'opzione di default: *Usa istruzioni Sql*, e clicchiamo su *Avanti*. A questo punto possiamo immettere manualmente l'istruzione SQL, oppure possiamo utilizzare la finestra di dialogo generatore di query. Per scrivere manualmente l'istruzione SQL non si deve fare altro che scrivere nella casella di testo multilinea l'opportuna istruzione di selezione nel linguaggio SQL. Per avere un aiuto possiamo cliccare sul pulsante *Generatore di query*. La finestra di dialogo *Generatore di query*, appare molto simile a quella che si utilizza normalmente in Access. Il primo passo consiste nello scegliere le tabelle da cui dobbiamo estrarre i dati. Per la nostra applicazione, si deve:

- selezionare la tabella *Contatti* nella finestra di dialogo *Aggiungi tabella;*
- cliccare sul pulsante *Aggiungi;*
- infine cliccare sul pulsante *Chiudi* per visualizzare la finestra di dialogo *generatore di query* nella quale è stata aggiunta la tabella *Contatti*.

In questa finestra possiamo selezionare i campi che vogliamo visualizzare, in modo da generare automaticamente la corrispondente istruzione SQL. Nella generazione dell'istruzione SQL, ci possono essere alcune differenze con quella che avrebbe generato Access, in particolare notiamo come i nomi dei campi non sono preceduti dal nome della tabella. Le differenze si verificano poiché il *Generatore di query* di Visual Basic .NET implementa *ANSI SQL* in modo leggermente diverso da Microsoft Access. Il pulsante *Opzioni* avanzate permette di visualizzare la finestra di dialogo *Opzioni* di generazione SQL avanzate in cui è possibile specificare le istruzioni SQL aggiuntive da generare per inserire, aggiornare ed eliminare dati nel database. Cliccando ancora su *Avanti* approdiamo all'ultima schermata della procedura guidata che visualizza un elenco riassuntivo delle selezioni e delle informazioni inserite durante lo svolgimento della procedura. Portiamo a termine la creazione dei componenti *OleDbDataAdapter* e *OleDbConnection* cliccando sul pulsante *Fine*. In questo modo i due nuovi componenti appaiono nella parte inferiore del form al di fuori dell'area di Progettazione Windows Form, così come accade per tutti i controlli non visivi (ricordiamo che i controlli non visivi vengono detti componenti).

Fig. 2: ***Il dettaglio del nostro progetto.***

La Toolbox, chiamata anche casella degli strumenti, contiene gli oggetti che è possibile aggiungere ai form in fase di progettazione. Per visualizzarla si può:

- **Selezionare la voce di menu View/Toolbox**
- **Premere l'icona Toolbox sulla barra degli strumenti standard**
- **Premere i tasti Ctrl+Alt+X**

Nel generatore di query di VB i campi nella tabella Contatti vengono elencati in ordine alfabetico mentre nel Query Builder di Microsoft Access i campi sono elencati nell'ordine di definizione.

IL DATASET

Alla fine della procedura guidata abbiamo ottenuto un componente che permette la connessione al database ed un componente che ci permette di recuperare i dati, il passo successivo consiste nel creare un nuovo componente per contenere i dati recuperati, il Dataset. Per creare un dataset è sufficiente selezionare il controllo dalla scheda dati nella Toolbox, ed inserirlo nel form, per poi impostare le opportune proprietà. Ancora più semplicemente, si può selezionare la voce di menu *Dati->Genera Dataset* in modo da aprire la corrispondente finestra di dialogo. La finestra contiene tre sezioni:

- Nella prima sezione è possibile selezionare un Dataset esistente nel progetto, oppure crearne uno nuovo. Come potete notare l'opzione *Esistente* è disabilitata poiché il progetto non contiene nessun Dataset, mentre l'opzione *Nuovo* è selezionata e mostra un nome di default che, nel nostro caso, possiamo accettare poiché andremo ad utilizzare un solo Dataset. Naturalmente in form con diversi dataset è opportuno assegnare nomi più significativi.
- Nella seconda sezione viene visualizzato l'elenco delle tabelle dalle quali il componente *OleDbDataAdapter* recupererà i dati. Anche in questo caso accettiamo i valori predefiniti.
- Nella terza sezione è presente una casella di controllo che permette di indicare se si vuole aggiungere un'istanza di DataSet alla finestra di progettazione. Contrassegnando questa casella di controllo, VB genera in automatico il codice necessario per dichiarare ed inizializzare il DataSet. Naturalmente questa scelta riduce la quantità di codice da aggiungere, quindi accettiamo anche questo valore predefinito.

Cliccando sul pulsante *OK* il componente *DataSet* viene generato ed aggiunto al progetto.

VISUALIZZARE I DATI

Dopo aver impostato tutti i componenti di accesso ai dati, dobbiamo scrivere il codice necessario per popolare il componente DataSet. Il metodo da utilizzare è il metodo *Fill* del componente *OleDbDataAdapter,* che utilizzerà l'istruzione SQL di selezione, per interrogare il database ed aggiungere righe di dati al componente DataSet. Nell'evento *Load* di Form1 possiamo scrivere semplicemente:

```
Private Sub Form1_Load(ByVal sender
                    As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    OleDbDataAdapter1.Fill(DataSet11)
End Sub
```

In questo modo il codice viene eseguito al caricamento del form popolando con i dati della tabella *Contatti* il componente DataSet. L'ultima operazione da compiere per visualizzare i dati contenuti nel dataset nel controllo *Datagrid,* è quella di impostarne le proprietà *DataSource* e *DataMember.* Per questo si deve:

- Selezionare il controllo *DataGrid* e visualizzare la finestra delle proprietà.
- Cliccare sull'elenco a discesa accanto alla proprietà *DataSource* e selezionare *DataSet11* (il nome del dataset appena creato).
- Cliccare sull'elenco a discesa accanto alla proprietà *DataMember* e selezionare la tabella contatti (l'unica presente nel dataset).

NOTA

Nella finestra di dialogo Proprietà di Data Link sono presenti due schede ulteriori. Nella scheda Avanzate, sono visualizzate le informazioni specifiche al provider scelto nella scheda Provider. Ad esempio per un database di Access, la sezione impostazioni di rete è disabilitata ma fornisce informazioni sul modo in cui la connessione comunica con il database. La sezione Altre fornisce informazioni su come deve essere stabilita la connessione con l'origine dati. Ad esempio L'opzione Timeout di connessione è disabilitata ma specifica la quantità di tempo di attesa della connessione mentre cerca di connettersi all'origine dati.
Anche nella scheda Tutte le proprietà, sono visualizzate le informazioni specifiche per il provider scelto nella scheda Provider. In queste due schede si possono accettare tutti i valori predefiniti.

A questo punto, in fase di progettazione, il *DataGrid* si presenterà vuoto con le intestazioni nelle colonne, non ci resta che eseguire il progetto per visualizzare tutti i dati presenti nella tabella contatti. Per completare l'applicazione, possiamo inserire un *CommandButton* che ci permetta di rendere permanenti sul database le eventuali modifiche effettuate nel *DataGrid.* Per questo è sufficiente utilizzare il metodo *Update* del componente *OleDbDataAdapter.*

```
Private Sub ButtonAggiorna_Click(ByVal sender As
            System.Object, ByVal e As System.EventArgs)
                        Handles ButtonAggiorna.Click
    OleDbDataAdapter1.Update(DataSet11)
End Sub
```

NOTA

Per curiosare sul codice scritto in automatico da VB, alla fine delle varie procedure guidate descritte nell'articolo, potete cliccare sul segno più (+) accanto alla sezione: Codice generato da progettazione Windows Form

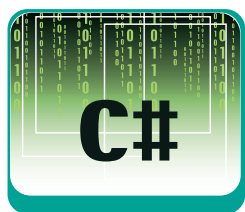
CONCLUSIONI

Con questo articolo abbiamo portato a termine il nostro viaggio all'interno delle modalità di accesso ai database, offerti da Visual Basic .NET.
Diamo appuntamento ai prossimi articoli con sempre nuovi argomenti.

Luigi Buono

Lettura e scrittura dei file

Input/Output parte seconda

Dopo aver introdotto l'impiego degli stream per lo scambio di dati ed informazioni, in questa lezione ci occuperemo dell'accesso ai file in lettura e scrittura, nelle diverse modalità possibili in .NET.

C# e .NET mettono a disposizione del programmatore diversi strumenti per la manipolazione dei file. Per il sistema operativo tutti i file sono costituiti da semplici sequenze di byte. Sono i singoli software che devono distinguere il contenuto e la tipologia di ogni file. Per aiutarci nella stesura del codice, il framework .NET mette a nostra disposizione differenti livelli di astrazione, che permettono di trattare i file sia come sequenze di byte che come sequenze di caratteri.
In questa lezione saranno esaminati entrambi i metodi di lettura e scrittura dei file.

APERTURA E CHIUSURA DI UN FILE

La lettura e la scrittura di un file avviene attraverso oggetti di tipo *System.IO.FileStream*. La classe *FileStream* definisce diversi costruttori, tra i quali il più interessante, al momento, è il seguente:

```
FileStream(string nomeFile, FileMode modo)
```

L'argomento *nomeFile*, si capisce, indica il percorso (assoluto o relativo) del file che si intende aprire; *modo* serve invece per specificare la modalità di apertura del file. Tale modalità di apertura va definita servendosi dell'enumerazione *FileMode*.
I valori possibili sono:

- **FileMode.Append.** Una modalità per accodare nuovi byte alla fine di un file esistente.
- **FileMode.Create.** Creare un nuovo file, ed eventualmente sovrascrivere un file omonimo già esistente.
- **FileMode.CreateNew.** Creare un nuovo file.
- **FileMode.Open.** Aprire un file già esistente.
- **FileMode.OpenOrCreate.** Aprire un file, se questo già esiste, oppure per crearne uno nuovo.
- **FileMode.Truncate.** Aprire un file già esistente cancellando tutto il contenuto eventualmente già presente al suo interno.

L'apertura di un file è un'operazione rischiosa, nel senso che è possibile incappare in una situazione anomala. Per questo motivo il costruttore di *FileStream* può propagare diversi tipi di eccezione.
Le più ricorrenti sono:

- **System.IO.FileNotFoundException.** Questa eccezione viene lanciata quando si cerca di interagire con un file che dovrebbe già esistere, ma che non è possibile trovare alla posizione indicata.
- **System.IO.IOException.** Lanciata al riscontrarsi di un generico errore di Input/Output che non rende possibile l'accesso al file.
- **System.Security.SecurityException.** Lanciata quando l'utente non ha i permessi necessari per l'accesso al file nella modalità desiderata.

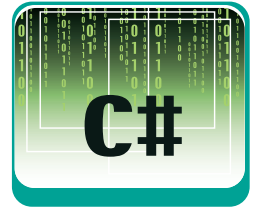
Una volta terminato l'utilizzo del file, è bene rilasciare le risorse occupate con una chiamata al metodo *Close()* di *FileStream*. In questo modo il file verrà chiuso in maniera regolare. Ecco un primo esempio che sintetizza un buono schema per il trattamento dei file:

```
using System.IO;
class Test {
  public static void Main() {
    FileStream fileStream = null;
    try {
      fileStream = new FileStream("test.dat", FileMode.Open);
      // Trattamento del file.
    } catch (FileNotFoundException e) {
      System.Console.WriteLine("Impossibile trovare il file");
    } catch (System.Security.SecurityException e) {
      System.Console.WriteLine("Non si dispone dei
            permessi necessari per l'accesso al file" );
    } catch (IOException e) {
      System.Console.WriteLine("Impossibile aprire il file");
    } finally {
      if (fileStream != null) try {
        fileStream.Close();
```

BIBLIOGRAFIA

• GUIDA A C#
Herbert Schildt
(McGraw-Hill)
ISBN 88-386-4264-8
2002

• INTRODUZIONE A C#
Eric Gunnerson
(Mondadori Informatica)
ISBN 88-8331-185-X
2001

• C# GUIDA PER LO SVILUPPATORE
Simon Robinson e altri
(Hoepli)
ISBN 88-203-2962-X
2001

```
      } catch (IOException) {}
    }
  }
}
```

Questo programma tenta l'apertura del file *test.dat,* gestendo le eccezioni sopra segnalate. Nel blocco *finally* il file viene chiuso. Questo è un ottimo schema di partenza per la progettazione di codici che realizzano l'accesso ai file. Con il costruttore di *FileStream* appena esaminato i file vengono aperti simultaneamente sia in lettura sia in scrittura. Spesso e volentieri, ad ogni modo, è possibile sapere a priori se il file dovrà essere solo letto o soltanto scritto. In questo caso è bene utilizzare un altro fra i costruttori di *FileStream,* strutturato al seguente modo:

```
FileStream(string nomeFile, FileMode modo, FileAccess
                                          tipoAccesso)
```

Con il terzo argomento *tipoAccesso* è possibile specificare in che maniera il file debba essere aperto. I valori possibili, quelli dell'enumerazione *FileAccess,* sono:

- **FileAccess.Read.** Accesso in sola lettura.
- **FileAccess.Write.** Accesso in sola scrittura.
- **FileAccess.ReadWrite.** Accesso in lettura e scrittura.

Ad esempio, l'accesso in sola lettura al file *test.dat* potrà essere eseguito scrivendo:

```
fileStream = new FileStream("test.dat",
                  FileMode.Open, FileAccess.Read);
```

I/O DI UN FILE COME SEQUENZA DI BYTE

Una volta stabilito il canale di comunicazione con un file, diventa possibile leggere e scrivere delle sequenze di byte. La lettura di un singolo byte avviene attraverso il metodo:

```
int ReadByte()
```

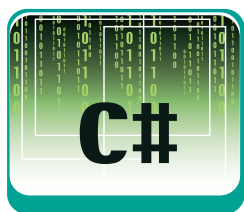
ReadByte() restituisce il valore letto sotto forma di numero *int.* Se è stata raggiunta la fine del file il valore restituito sarà -1. In caso di errore il metodo propaga una *IOException.* Il seguente programma dimostrativo legge, uno ad uno, i byte che compongono il file *test.dat,* stampandoli in output:

```
using System.IO;
class Test {
  public static void Main() {
    FileStream fileStream = null;
    try {
      fileStream = new FileStream("test.dat",
                        FileMode.Open, FileAccess.Read);
      int b;
      while ((b = fileStream.ReadByte()) != -1) {
        System.Console.WriteLine(b);}
    } catch (FileNotFoundException e) {
      System.Console.WriteLine("Impossibile trovare il file");
    } catch (System.Security.SecurityException e) {
      System.Console.WriteLine("Non si dispone dei
                    permessi necessari per l'accesso al file");
    } catch (IOException e) {
      System.Console.WriteLine("Errore I/O");
    } finally {
      if (fileStream != null) try {
        fileStream.Close();
      } catch (IOException) {}
    }
  }
}
```

Spesso e volentieri, ad ogni modo, anziché leggere un singolo byte alla volta, si desidera acquisire in un colpo unico un blocco di byte. Il metodo utile per compiere questa operazione è:

```
int Read(byte[] buffer, int offset, int length)
```

Il metodo tenta la lettura di *length byte* dal canale di comunicazione stabilito. I byte letti vengono copiati all'interno dell'array buffer, a partire dalla sua posizione offset. Il valore restituito rappresenta il numero di byte effettivamente letti (che può essere inferiore a *length*). Se non è stato possibile leggere alcun byte perché la fine del file è stata raggiunta, allora il valore restituito sarà 0 (attenzione, quindi, non è come *ReadByte()* che in tal caso ritorna -1). Come nel caso precedente, anche il metodo *Read()* può propagare una *IOException.* Il seguente programma è analogo al precedente, con la differenza dell'impiego del metodo *Read()* e di un buffer di byte al posto del più semplice *ReadByte():*

```
using System.IO;
class Test {
  public static void Main() {
    FileStream fileStream = null;
    try {
      fileStream = new FileStream("test.dat",
                        FileMode.Open, FileAccess.Read);
      byte[] buffer = new byte[1024];
      int letti;
      while ((letti = fileStream.Read(buffer, 0,
                                  buffer.Length)) > 0) {
        for (int i = 0; i < letti; i++) {
          System.Console.WriteLine(buffer[i]);}
        }
    } catch (FileNotFoundException e) {
```

```
      System.Console.WriteLine("Impossibile trovare il file");
    } catch (System.Security.SecurityException e) {
      System.Console.WriteLine("Non si dispone dei
                    permessi necessari per l'accesso al file");
    }
    catch (IOException e) {
      System.Console.WriteLine("Errore I/O");
    }
    finally {
      if (fileStream != null) try {
        fileStream.Close();
      } catch (IOException) {}
    }
  }
}
```

Nel programma è stato messo in uso un buffer da 1 *KB (=1024 byte)*. Questa è solitamente una dimensione ottimale per la lettura di un file con gli attuali sistemi operativi. Leggere un blocco di byte è molto più efficiente che leggere un byte alla volta. Pertanto la tecnica di lettura attraverso *Read()* ed un buffer è generalmente da preferirsi rispetto a *ReadByte()*, salvo casi particolari. La scrittura è un'operazione specularmente simile alla lettura. Il metodo di scrittura più semplice è costituito da:

```
void WriteByte(byte b)
```

WriteByte() scrive sul file il valore *b*. Il metodo può sollevare una *IOException*. La scrittura può avvenire anche in maniera più compatta attraverso il metodo:

```
void Write(byte[] buffer, int offset, int length)
```

Il metodo scrive sul file *length byte* estratti da buffer, a partire dalla sua posizione *offset*. Può essere sollevata una *IOException*, in caso di errore. Spesso e volentieri il sistema operativo ottimizza le operazioni di scrittura servendosi di un buffer interno, trasparente nei confronti del programmatore. Per questo motivo, dopo una chiamata a *WriteByte()* o a *Write()*, non è sicuro che i byte inviati siano già stati depositati su disco. Può darsi che il sistema li stia conservando in attesa di altri dati, per scrivere tutto in un colpo solo. La scrittura effettiva può essere esplicitamente comandata in qualsiasi momento con il metodo:

```
void Flush()
```

Ad ogni modo la chiusura di un file tramite *Close()* esegue automaticamente l'effettiva scrittura dei byte eventualmente rimasti sospesi nel buffer del sistema. Per questo motivo, nella maggior parte dei casi, non c'è bisogno di appellarsi a *Flush()*: basta chiudere regolarmente il file, come comunque andrebbe fatto. La seguente applicazione presenta un caso di studio interessante, combinando lettura e scrittura dei file:

```
using System.IO;
class Copy {
  public static void Main(string[] files) {
    if (files.Length != 2) {
      System.Console.WriteLine(
        "Specificare il file sorgente e quello di destinazione");
      return; }
    FileStream readStream = null;
    FileStream writeStream = null;
    try {
      readStream = new FileStream(files[0],
                         FileMode.Open, FileAccess.Read);
      writeStream = new FileStream(files[1],
                         FileMode.Create, FileAccess.Write);
      byte[] buffer = new byte[1024];
      int letti;
      while ((letti = readStream.Read(buffer, 0,
                                  buffer.Length)) > 0) {
        writeStream.Write(buffer, 0, letti);
      }
    } catch (FileNotFoundException e) {
      System.Console.WriteLine("Impossibile trovare il file");
    } catch (System.Security.SecurityException e)
    {
      System.Console.WriteLine("Non si dispone dei
                    permessi necessari per l'accesso ai file");
    } catch (IOException e)
    {
      System.Console.WriteLine("Errore I/O");
    } finally {
      if (writeStream != null) try {
        writeStream.Close();
      } catch (IOException) {}
      if (readStream != null) try {
        readStream.Close();
      } catch (IOException) {}
    }
  }
}
```

L'applicazione mostrata è un'imitazione del comando *COPY* del DOS: esegue la copia di un file.
Lo potete richiamare da riga di comando digitando:

```
NomeProgramma <file_di_origine> <file_di_copia>
```

CONCLUSIONI

Termina qui la lezione dedicata alla lettura e alla scrittura dei file. Il mese prossimo ci dedicheremo ad altri argomenti collegati all'impiego degli stream e del sistema di Input/Output di C#.

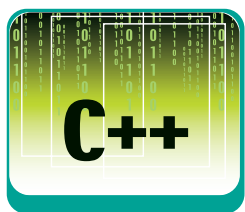
Carlo Pelliccia

CONTATTA L'AUTORE

Se desideri contattare l'autore di questo articolo scrivi a carlo.pelliccia@ioprogrammo.it

Tecniche e trucchi per migliorare le prestazioni dei nostri programmi

Ottimizzazione del codice

Abbiamo imparato a programmare usando gli oggetti, ed abbiamo "spulciato" la libreria standard del C++, e le STL: è ora di imparare qualche trucco per rendere più efficienti i nostri programmi.

A chi volesse approfondire la sua conoscenza sulle librerie standard del C++, consigliamo il validissimo libro

• THINKING IN C++ - 2ND ED. - VOLUME 2 *Bruce Eckel* e *Chuck Allison*

che rappresenta sicuramente un ottimo riferimento per i programmatori più avanzati (o aspiranti tali) ed è oltretutto disponibile gratuitamente per il download, partendo dall'indirizzo: http://www.mindview.net/Books/TICPP/ThinkingInCPP2e.html

Abbiamo appena finito di preparare il gioco del secolo: trama avvincente, grafica spettacolare, giocabilità imbattibile. Stiamo per diventare ricchi! Questo almeno finché non andiamo a stilare il profilo hardware minimo necessario a far funzionare il tutto... e lì ci rendiamo conto che dobbiamo almeno ridimensionare la lunghezza del nostro yacht, relegando il nostro videogame ad una nicchia di utenti che hanno un PC paragonabile ai sistemi di calcolo della NASA.
Questa spiacevole situazione si svolge probabilmente almeno una volta per ogni progetto: prima o poi tutti debbono fare i conti con i vincoli nascosti nello sviluppo di un software: quelli prestazionali.
Per ottimizzazione del codice si intende la modifica di alcune sue parti, con l'intento di diminuire le risorse necessarie al corretto funzionamento del nostro programma. Le risorse che un programma consuma sono essenzialmente di due tipi:

- il tempo di utilizzo della CPU
- la memoria occupata.

Di solito queste due sono correlate strettamente, ad esempio: per sprecare meno tempo di CPU, dobbiamo spesso ricorrere a strutture dati più complesse ed estese, tali strutture però, essendo più estese, occupano più memoria e quindi sprecano qualcosa del tempo della CPU in allocazione.
Si intuisce che per un programma ci sarà un compromesso ottimo tra quantità di memoria usata e tempo di CPU speso, e il compito dell'ottimizzazione è avvicinarci quanto più possibile a tale compromesso.

"REGOLE" GENERALI

Nell'ambito in cui ci stiamo muovendo, non esistono delle vere e proprie regole, dei teoremi dimostrabili la cui validità sia assoluta: esistono solo le possibili verifiche che possiamo fare con il cronometro (e un analizzatore di risorse) alla mano. Esistono comunque alcuni principi che risultano sempre validi, ad esempio:

A. La velocità di un programma che non funziona è irrilevante.

Possiamo considerarla la "prima legge dell'ottimizzazione". Quando un programma non funziona, non serve a niente, e se l'ottimizzazione lo espone a troppi rischi allora forse è il caso di innalzare la potenza dell'hardware sul quale il programma gira (sempre che questo upgrade sia una cosa possibile), mantenendo il programma non ottimizzato ma funzionante. In altre parole, va sempre considerata la possibilità di un soluzione hardware a fianco all'ottimizzazione del software.

B. La fase di ottimizzazione deve essere posticipata il più possibile.

Il motivo è semplice: quando si ottimizza, di solito si va contro ciò cui finora ci siamo raccomandati, si diminuisce la leggibilità del codice, si compiono operazioni "rischiose", spesso rinunciando alla gestione delle eccezioni (che possono talora indurre un sovraccarico di lavoro non indifferente per la CPU, anche dell'ordine del 10%) e si spera che tutto vada sempre bene e che errori non ce ne siano mai in certi passaggi critici. Detto in altre parole, meglio un programma che funziona ra-

gionevolmente piano che uno che va rapidissimamente in crash. Quindi un approccio ingegneristico al problema prevede dapprima la stesura di un programma che funzioni bene con prestazioni decenti, poi una fase di ottimizzazione che verta a migliorare tali prestazioni senza pregiudicarne la correttezza.

C. Il miglior compilatore è tra le nostre orecchie.

I compilatori, di pari passo con l'evolversi degli strumenti informatici, stanno diventando sempre più "intelligenti", tuttavia non sono ancora in grado di capire sempre cosa vogliamo: solo noi sappiamo realmente cosa stiamo scrivendo. Un compilatore è solamente uno strumento automatico, e alcuni tipi di errori non è in grado di rilevarli, né è in grado di capire il funzionamento generale di un algoritmo in modo da poterlo ottimizzare.

D. Si ottimizzano i programmi troppo ingombranti o troppo lenti.

In altre parole l'ottimizzazione deve condurre a dei benefici tangibili e non deve essere fine a se stessa. Mai ottimizzare all'estremo, tranne quando le prestazioni di un software sono l'aspetto fondamentale dello stesso. Ad esempio è inutile ottimizzare un programma velocizzandone l'esecuzione del 50% se quel programma funziona e viene usato poche volte in un anno: il tempo speso nella sua ottimizzazione verrà ammortizzato solo dopo diversi decenni di esecuzione. Ha invece senso ottimizzare anche solo del 10% un software usato per otto ore al giorno, tutti i giorni: se anche la sua ottimizzazione richiedesse duecento ore, tuttavia usandolo per otto mesi già ci saremo rifatti dell'investimento.

Come si evince facilmente da queste "regole", ottimizzare non è facile, commettere errori o dimenticare di evitarne altri è sempre possibile e in alcuni casi può diventare anche abbastanza probabile: il principale componente dell'ottimizzazione è l'esperienza, perché è attraverso l'esperienza che il programmatore acquisisce il buonsenso necessario ad effettuare accorte ottimizzazioni, a non esagerare.

ERRORI COMUNI

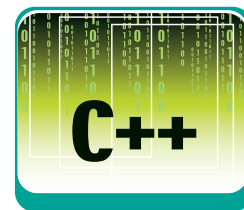
Facciamo innanzi tutto una carrellata di quelli che sono gli errori più comuni quando si vuole ottimizzare un software. Sicuramente l'errore più comune che viene compiuto è quello di fare delle assunzioni sbagliate (non nel senso di assumere programmatori poco bravi, anche se assumere programmatori competenti non è una cattiva idea!), cioè delle ipotesi sulle prestazioni del nostro software. Ad esempio, pensare che sia scontato che una funzione venga eseguita più rapidamente di un'altra, oppure che sia certo che non è un particolare tipo di dato la causa dei nostri problemi. Quando si parla di prestazioni gli unici che hanno voce in capitolo sono il cronometro e il misuratore di risorse utilizzate: mai supporre nulla, misurare tutto (la misura è il cuore dell'ingegneria). Segue a ruota il pensare che un codice breve è eseguito più rapidamente di un codice lungo. Un esempio banale che evidenzia l'erroneità di tale idea è il loop-unrolling, che consiste nell'esplicitare quando possibile i cicli (come i *while* o i *for*); ad esempio:

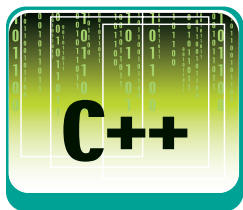
```
int N = 5;
for(int i=0;i<N;i++)
   cout << i << "!\n";
```

effettuando il loop-unrolling di questo ciclo *for* si avrà:

```
cout << "1!\n";
cout << "2!\n";
cout << "3!\n";
cout << "4!\n";
cout << "5!\n";
```

che ottiene gli stessi risultati, ma, ad esempio, richiede due variabili di meno da dichiarare e modificare e ben 5 confronti in meno (l'esempio è banale, ma si pensi ad oggetti ben più complessi e si comprenderà che il costo di una dichiarazione o di un accesso può essere elevato). Il loop-unrolling non è sempre possibile, ma quando lo è rende di solito il codice più efficiente della sua controparte non esplicitata. In altri termini, la rapidità di esecuzione non è legata alla lunghezza del codice. Altro errore comune è di tipo metodologico: non tutte le ottimizzazioni possono compiersi all'atto della scrittura del codice. Ottimizzare ogni tratto di codice nel momento in cui viene scritto non è mai una scelta saggia, anche se in alcuni casi è possibile ottimizzare codice mentre lo si scrive. In generale, come anticipato, l'ottimizzazione è sempre l'ultimo passo: tutti i passi precedenti devono essere finalizzati all'ottenimento di un software che funzioni bene, senza curarsi (troppo) delle prestazioni. Imparare ad evitare questi errori comuni sa-

SUL WEB

Se volete dare un'occhiata a una reference delle funzioni standard del C per la manipolazione di stringhe (o meglio: di array di caratteri terminati da "\0" :-) consultate l'indirizzo:
http://www.cplusplus.com/ref/cstring/

Online è inoltre disponibile l'utilissimo "C++ Annotations" all'URL:
http://www.icce.rug.nl/documents/
che merita di essere visto (e letto) almeno una volta.

rebbe già un'ottima linea guida, ma non basterebbe: dobbiamo infatti considerare anche altri aspetti dell'ottimizzazione, magari inaspettati.

PROGETTARE È GIÀ OTTIMIZZARE

Quando si ottimizza, lo si deve fare con un obiettivo preciso (e ragionevole) in mente: prima di scrivere codice, dobbiamo decidere su quali prestazioni vogliamo attestarci. Ad esempio, potremmo volere un output a schermo di 100 elementi al secondo, oppure un tempo di elaborazione di un calcolo non superiore al minuto. Una volta che avremo deciso ciò, inizieremo ad ottimizzare il codice, e lo faremo dopo una accurata valutazione di quali siano i colli di bottiglia, tenendo sempre un occhio verso la nostra meta finale. L'ottimizzazione di un software però non è solo un processo che segue la stesura del codice funzionante: come anticipato, alcune cose possono essere fatte anche durante la stesura, ma soprattutto prima. Quando dobbiamo programmare il nostro software, infatti, dobbiamo sempre perdere un bel po' di tempo nel progetto ("perdere" è in questo caso il sinonimo di "guadagnare": nel processo di programmazione di un software, il classico dilemma "meglio l'uovo oggi o la gallina domani?" ha come risposta "la gallina (il software) domani!"): una accurata scelta dell'architettura (ad esempio, per la nostra rubrica telefonica è meglio un database convenzionale o appoggiarsi a dei semplici file di testo?), una attenta selezione delle strutture dati (ad esempio, meglio un *intero* o un *double* per i nostri fini? si immagini di dover valutare una analoga scelta con strutture dati ben più complesse), una scelta ottimale per gli algoritmi, sono tutte tecniche di ottimizzazione preventiva di un software. Senza contare il costo che ha poi una scelta progettuale sbagliata da correggere alla fine del processo di sviluppo del software, il che da solo basta a giustificare il noto principio del "40-20-40", uno dei primi che si apprendono addentrandosi nell'ingegneria del software: la suddivisione ottimale dei tempi di sviluppo prevede il 40% del tempo speso in progettazione, 20% in codifica, 40% in test. Un altro consiglio indubbiamente da seguire è quello di valutare le prestazioni di versioni stabili e quasi finali del software: è inutile valutare le prestazioni di un codice solo abbozzato, oppure del codice con annesse le istruzioni di debug. Dette queste cose, non rimane che dire che vi sono tre tipi di ottimizzazione del codice corrispondenti ad altrettante fasi in cui sono applicabili:

- ottimizzazione preventiva;
- ottimizzazione durante la codifica;
- ottimizzazione finale.

Della prima abbiamo appena accennato i metodi e gli scopi. Delle ottimizzazioni durante la stesura del codice possiamo dire che di solito sono azioni non strutturali che però possono incidere ampiamente sulle prestazioni, e che di solito incrementano la leggibilità del codice e la sua manutenibilità. Le ottimizzazioni finali sono invece quelle di cui si parlava all'inizio: si ottimizza in maniera decisiva alla fine dello sviluppo del software, e un esempio estremo è la scelta dei comandi più opportuni da dare al nostro compilatore.

ALCUNI CONSIGLI PRATICI

I consigli teorici visti sinora devono essere la falsariga da tenere in mente ogniqualvolta ci si ritrova a dovere ottimizzare il proprio codice. Tuttavia per "sporcarci le mani" qualche trucco dobbiamo conoscerlo e, come vedremo, mentre alcuni sono abbastanza intuibili, altri vanno contro il senso comune del programmatore medio. Vediamone alcuni:

1. Usare le STL.

Se avete seguito le precedenti puntate di questo corso già saprete delle grandi qualità di questa libreria software. Una quantità davvero sorprendente di codice già scritto (dai contenitori agli algoritmi che operano su di essi) è disponibile per i nostri scopi di programmazione. Si raccomanda in particolare di preferire il tipo *string* ogniqualvolta si debbano compiere operazioni complesse su array di caratteri (ricerca di un carattere, copia di porzioni di stringa ecc.).
Le string, oltre a essere più veloci, ci sollevano anche dall'incombenza di dovere gestire il terminatore di fine stringa *"\0"*. In generale comunque le STL effettuano tutte le operazioni più comuni (ad esempio ordinamento di un array, ricerca di un elemento ecc.) utilizzando algoritmi che prevedono la soluzione ottima teorica del particolare problema in esame e sono quindi ottimizzati "in partenza".

2. Passare oggetti per riferimento.

Nella scrittura di una funzione che prenda come argomento un oggetto (sia esso di una classe definita da noi oppure no) è preferibile, ai fini dell'efficienza di esecuzione, utilizzare il passaggio

I seguenti libri sono stati usati come base per la puntata corrente:

[1] ZEN OF CODE OPTIMIZATION: THE ULTIMATE GUIDE TO WRITING SOFTWARE THAT PUSHES PCS TO THE LIMIT
Michael Abrash

[2] C++ OPTIMIZATION STRATEGIES AND TECHNIQUES
Pete Isensee
reperibile all'indirizzo Web: http://www.tantalon.com/pete/cppopt/main.htm e punto di riferimento per chiunque volesse approfondire l'argomento. Ne consigliamo vivamente la lettura.

[3] OPTIMIZING C++ - THE WWW VERSION
Steve Heller
ottimo libro consultabile online e stampabile. Lo potete leggere all'indirizzo Web: http://www.steveheller.com/opt/.

per riferimento anziché passare il puntatore all'oggetto. La funzione:

```
void prova(MioOggetto& o) {...}
```

è quindi da preferirsi a:

```
void prova(MioOggetto* o) {...}
```

cosa che può sembrare strana; visto che in entrambi i casi è passato un indirizzo di memoria a tutti gli effetti, (generalmente una stringa di 32 bit). I motivi dell'efficienza stanno da altre parti: ad esempio non dovremo occuparci, nel codice della funzione, di gestire il caso in cui sia passato un puntatore *NULL*, o che comunque non referenzia nessuna zona di memoria valida. Inoltre, l'utilizzo del puntatore impedisce al compilatore di sapere se, ad esempio, due puntatori si riferiscono alla stessa zona di memoria, o altre informazioni di questo tipo, che possono permettere delle ottimizzazioni particolari a tempo di compilazione. Ricordiamo invece che è bene evitare come la peste il passaggio per valore

```
void prova(MioOggetto o) {...}
```

in quanto questo comporta la chiamata del costruttore di copia del particolare oggetto e, appunto, l'istanziazione di un altro oggetto uguale, con conseguente utilizzo di memoria e tempo aggiuntivi. Il passaggio per valore dovrebbe essere preso in considerazione soltanto come unica soluzione per mantenere la correttezza del codice (che, come già detto, resta la cosa più importante).

3. Prevedere una costruzione in due fasi.
Costruire un oggetto in due fasi significa scrivere un costruttore che sia il più possibile "leggero" e istanzi soltanto il minimo indispensabile per non provocare errori. Oltre a questo, prevedere una funzione che faccia la costruzione "complessa" soltanto quando questa è richiesta. Per fare un esempio (paradossale) supponiamo di avere un oggetto Automobile con i classici campi *"Modello"* e *"Targa"*, supponiamo però che questi campi debbano essere costruiti accedendo a un database remoto.
Il costruttore *"standard"* sarebbe:

```
Automobile() {
        Targa = GetTargaFromDB();
        Modello = GetModelloFromDB();}
```

dove le funzioni *GetXXX()* andrebbero a sprecare molte risorse per una semplice istanziazione dell'oggetto. Una possibile soluzione è quella di prevedere un costruttore più semplice, e una funzione separata che si occupi di effettuare la parte "importante" del lavoro, cioè l'accesso al database:

```
//costruttore che evita codice complesso
Automobile() {
        Targa = "313";
        Modello = "Auto di Paperino";}
//funzione che "costruisce" effettivamente l'oggetto
GetDataFromDB() {
        Targa = GetTargaFromDB();
        Modello = GetModelloFromDB();
}
```

Questo accorgimento può fare risparmiare molto oppure essere ininfluente: molto dipende dal grado di complessità del codice di inizializzazione e dalle operazioni che sono compiute prima della costruzione con dati significativi.

4. Utilizzare *printf()* anziché *cout*.
Questo è uno dei consigli che vanno controcorrente ed effettivamente ci sentiamo di proporre questa ottimizzazione solamente nei casi dove ce ne sia bisogno assoluto e non si possa agire su altri parametri. I vantaggi dell'utilizzo degli stream (come *cout*) al posto di funzioni C-like sono stati spiegati più volte su queste pagine, tuttavia c'è da dire che scrivere

```
printf ("%c %d %f",'C',313,176.671);
```

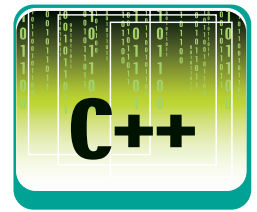
può risultare più veloce che utilizzare molte volte *operator<<()* in congiunzione con *cout:*

```
cout << 'a' << ' ' << 313 << ' ' << 176.671;
```

In taluni casi può essere anche più leggibile a livello di codice sorgente (ad esempio se si usano di frequente i manipolatori di flusso). Inutile dire che l'output prodotto è il medesimo, ma questo è un requisito fondamentale della fase di ottimizzazione.

CONCLUSIONI

In questa puntata abbiamo iniziato a conoscere il mondo dell'ottimizzazione del codice scritto in C++. Dopo una breve introduzione teorica abbiamo visto un assaggio di quelli che sono i "trucchi" che il programmatore smaliziato deve apprendere per rendere le sue "creature" più efficienti e veloci. Nella prossima puntata continueremo con l'argomento analizzando altre tecniche: non mancate!

Alfredo Marroccelli, Marco Del Gobbo

Se hai suggerimenti, critiche, dubbi o perplessità sugli argomenti trattati e vuoi proporle agli autori puoi scrivere agli indirizzi:

alfredo.marroccelli@ioprogrammo.it **e** marco.delgobbo@ioprogrammo.it

Questo contribuirà sicuramente a migliorare il lavoro di stesura delle prossime puntate.

Costruire applicazioni stabili e veloci

Un problema di memoria

Questo mese scoprirai alcune caratteristiche sorprendenti del linguaggio Java, che ti porteranno verso un mondo nuovo: la gestione della memoria.

Questa lezione sarà un po' più teorica del solito, senza esercizi. Non dimenticare però di giocare un po' con il codice per memorizzare bene questi concetti. Guarda questo programma:

```
class ChiamataConPrimitive {
  static void incrementa(int x) {
    x = x + 1; }
  public static void main(String[] args) {
    int x = 10;
    incrementa(x);
    System.out.println(x); }}
```

Secondo te, quale sarà il risultato?

UN RISULTATO INATTESO (FORSE)

Il programma *ChiamataConPrimitive* stampa il numero 10. Infatti il metodo *incrementa()* non ha incrementato la variabile originale, ma una sua copia (che, tanto per complicare la situazione, ho chiamato con lo stesso nome). Quando la variabile *x* viene passata al metodo, il sistema ne copia il valore nel parametro *x*. Per questo si dice che in Java i parametri primitivi vengono passati per copia. Poi il parametro *x* viene incrementato. Al termine del metodo il parametro *x* esce dal suo campo di visibilità e "muore". In tutto questo processo la variabile originaria *x* non è mai stata modificata, quindi vale ancora 10. Vediamo cosa succede se incapsuliamo la variabile in un oggetto:

```
class Numero {
  int num;
  Numero(int x) {
    num = x; }
}
```

La classe *Numero* ha un costruttore che assegna un valore iniziale al campo *num*. Ecco un programma simile al precedente, che però incrementa un oggetto *Numero* anziché un intero:

```
class ChiamataConOggetti {
  static void incrementa(Numero y) {
    y.num++; }
  public static void main(String[] args) {
    Numero x = new Numero(10);
    incrementa(x);
    System.out.println(x.num);} }
```

Cosa stamperà il programma *ChiamataConOggetti*?

LE COSE CAMBIANO

Sorpresa... Questa volta il risultato è 11. Quindi gli oggetti si comportano in modo diverso dalle variabili primitive. Non abbiamo passato al metodo *incrementa()* una copia dell'oggetto, ma l'oggetto "in persona", che viene temporaneamente rinominato *y*. Dopo che il metodo incrementa il valore del campo *num*, ritroviamo il valore modificato anche nel nostro oggetto originale. Quindi gli oggetti di Java, a differenza delle variabili primitive, non vengono passati per copia. Nel linguaggio dell'informatica possiamo dire che Java passa gli oggetti ai metodi per riferimento. Questa regola si applica non solo al passaggio dei parametri, ma a qualsiasi assegnamento:

```
class Assegnamenti {
  public static void main(String[] args) {
    int x = 10;
    int y = x;
    y++;
    System.out.println(x);
```

NOTA

Sul CD allegato a questo numero di ioProgrammo troverai un file ZIP che contiene parte del codice di questo mese.

```
Numero n = new Numero(10);
Numero n2 = n;
n2.num++;
System.out.println(n.num); } }
```

Cosa stampa il programma *Assegnamenti?* Forse avrai intuito che il risultato della prima stampa è 10 e quello della seconda è 11. Quando assegni una variabile primitiva ad un'altra variabile primitiva, Java ne copia il valore. Quando assegni un oggetto ad un altro oggetto, invece, succede qualcosa di strano:

```
Numero n = new Numero(10);  // creo il numero n
Numero n2 = n; // definisco un numero n2 uguale a n
// ora n ed n2 sono *lo stesso oggetto*!
n2.num++; // incremento n2, e quindi anche n
System.out.println(n.num);  // n e' stato incrementato
                                                e vale 11
```

Questo fenomeno apparentemente bizzarro si chiama aliasing. Nel nostro caso *n2* è un alias di *n*, cioè un altro nome per lo stesso oggetto. Questo fatto può apparire bizzarro. Per capirne i motivi dobbiamo aprire il cofano e guardare nel motore, al modo in cui Java rappresenta le variabili primitive e gli oggetti in memoria.

SOTTO IL COFANO

Cosa succede quando definisci una variabile?

```
int x = 10;
```

Probabilmente sai già che questa istruzione contiene in realtà due istruzioni separate: una *dichiarazione*, con la quale dici al compilatore che vuoi una variabile con un dato tipo e un dato nome; e un *assegnamento*, con il quale scrivi un valore nella variabile.

```
int x;   // definisci una variabile intera di nome x
x = 10;  // assegna il valore 10 alla variabile
```

La prima istruzione dice: *"voglio una variabile intera di nome x"*. Quindi Java riserva un pezzetto di memoria con le dimensioni necessarie e sufficienti a contenere un intero, cioè 8 byte (uno dei motivi per cui i programmi Java girano su diverse piattaforme è il fatto che i tipi primitivi hanno dimensioni predefinite su qualsiasi computer, cosa che non è garantita in un linguaggio come il C). La memoria prenotata da *x* viene "recintata" in modo che nessun altro oggetto o variabile possa invaderla e scriverci dentro. Si dice che il sistema alloca la memoria necessaria alla variabile *x*. Qual è il contenuto di questi 8 byte? Quando il sistema alloca la memoria, il contenuto è indefinito. Quei byte contengono garbage, spazzatura: forse avranno avuto un significato in passato, quando quella memoria era occupata da qualche altra variabile – ma ora sono solo bit a casaccio che potrebbero rappresentare qualsiasi valore intero. Per fortuna Java ci impedisce di allocare memoria e leggerne il contenuto senza prima averci scritto dentro qualcosa di sensato. Se la variabile in questione è il campo di un oggetto, il sistema la inizializza a *zero*. Se la variabile è definita in un metodo, il compilatore Java verifica che tu ci scriva dentro qualcosa prima di leggerne il valore. Nel nostro caso abbiamo inizializzato *x* con il valore 10. Quindi il sistema va negli 8 byte riservati ad *x* e ci scrive dentro la rappresentazione binaria del numero 10. Cosa succede quando assegni il valore di *x* ad un'altra variabile?

```
int y;
y = x;
```

La prima riga funziona esattamente come nel caso precedente: il sistema alloca la memoria per *y*. La seconda riga copia il valore contenuto dalla memoria occupata da *x* a quella occupata da *y*. Quindi *x* ed *y* sono due pezzetti di memoria separati, senza alcun rapporto. Contengono lo stesso valore. Ma se cambio il valore di uno dei due, l'altro resta invariato. Per le variabili primitive, quindi, non c'è nessun problema di aliasing. Vediamo ora cosa succede quando definiamo un oggetto:

```
Numero n = new Numero(10);
```

Anche questa istruzione nasconde una dichiarazione ed un assegnamento:

```
Numero n;
n = new Numero(10);
```

Il caso degli oggetti è un po' più complicato di quello delle variabili primitive, perché l'assegnamento include anche la creazione dell'oggetto. Quando il compilatore incontra la dichiarazione dell'oggetto, deve allocare memoria. Ma a differenza di quello che accade con le variabili primitive, il compilatore non sa quanta memoria allocare. Se l'oggetto fosse una stringa, ad esempio, quanto sarebbe lunga questa stringa? Forse questa stringa verrà calcolata con un algoritmo molto complicato, o verrà inserita dall'utente a tempo di esecuzione. Tutto quello che il compilatore può fare è riservare un minimo di memoria per l'informazione: "questo *è un oggetto che si chiama n*", e poi agganciare questa informazione all'oggetto vero e proprio, quando sarà disponibile.
Quindi il sistema alloca un piccolo spazio di memoria per un reference (un "riferimento") all'oggetto. Un *reference* somiglia ad una variabile: ha un nome, un tipo e occupa lo spazio di pochi byte. Ma a diffe-

OBIETTIVI DI QUESTA LEZIONE

Ti imbatterai nel problema dell'aliasing. Imparerai una cosa o due su come Java gestisce la memoria. Conoscerai gli oggetti immutabili e la parola chiave final.

REFERENCE E PUNTATORI

Qualcuno paragona i "reference" di Java ai puntatori del linguaggio C. Ma i puntatori sono in effetti semplici indirizzi in memoria, e permettono giochini pericolosi che in Java non sono possibili (ad esempio è possibile incrementare un puntatore per "puntare" alla cella di memoria successiva). Tutto quello che puoi fare con un reference è parlare con il suo oggetto o copiarlo in un altro reference.

renza di una variabile, un reference non contiene i dati veri e propri. È solo una piccola quantità di informazioni che deve essere "collegata" ad un oggetto. Un reference non ancora collegato ha un valore speciale che si chiama *null*, e vuol dire appunto *"nessun oggetto è collegato a questo reference"*. Quindi la dichiarazione non crea un oggetto, ma definisce un reference. La creazione dell'oggetto avviene nella seconda riga, quando chiamiamo il costruttore dell'oggetto con l'istruzione *new*. A questo punto il sistema alloca la memoria per l'oggetto e chiama il costruttore. Quando la costruzione è finita, l'istruzione *new* restituisce un "collegamento" all'oggetto che deve essere messo da qualche parte, altrimenti va perso. È come se il *new* gettasse una corda collegata all'oggetto, e se non raccogliamo subito questa corda l'oggetto va inesorabilmente alla deriva. Il modo più comune di recuperare questa corda è quello di collegarla subito ad un reference già pronto, ed è per questo che quasi sempre prima di un *new* abbiamo un assegnamento. Nel nostro caso colleghiamo l'oggetto neonato al reference n. Il compilatore verifica sempre che il tipo del reference e quello dell'oggetto siano compatibili, quindi non possiamo ad esempio assegnare un oggetto di tipo *String* ad un reference di tipo *Numero*. Da questo momento in poi, il reference è il nostro "guinzaglio" per l'oggetto. In Java non trattiamo mai direttamente con gli oggetti: tutto quello che facciamo (compreso chiamare un metodo o accedere a un campo) avviene tramite un reference. Quando "assegni" un oggetto ciò che viene copiato non è il valore dell'oggetto, ma solo il contenuto del reference.

```
Numero n2 = n; // copia il valore del reference n nel
                                            reference n2
```

Tutto questo spiega perché gli oggetti sono soggetti al fenomeno dell'aliasing. Quando copi un reference in un altro reference, entrambi puntano allo stesso oggetto. Lo stesso avviene quando passi l'oggetto come parametro ad un metodo: il reference del metodo punta allo stesso oggetto che gli hai passato da fuori. Quindi, in un certo senso, tutti i passaggi di parametri (e tutti gli assegnamenti) in Java avvengono per valore. Gli oggetti sono passati e assegnati per riferimento, perché ad essere passati o assegnati per valore sono in realtà i loro reference. Questa è in generale una cosa positiva. In alcune situazioni potresti desiderare che il metodo modifichi l'oggetto originale, e non una sua copia.

GLI EFFETTI COLLATERALI

Il metodo ideale legge i suoi argomenti, fa qualche calcolo e restituisce un valore senza cambiare il valore degli argomenti. Ad esempio:

```
public void stampa(Numero n) {
  System.out.println(n);}
public Numero somma(Numero op1, Numero op2) {
  return new Numero(op1.num + op2.num);}
```

Il primo metodo stampa semplicemente il valore di un *Numero*. Il secondo somma due *Numeri*, ma non modifica il valore degli operandi: semplicemente usa i valori dei due oggetti per costruire un nuovo *Numero* (il reference al nuovo oggetto viene restituito al client, e sarà compito del client assegnarlo ad un reference). Quando un metodo non modifica il valore dei propri parametri, si dice che non ha effetti collaterali. Sarebbe bene che la gran parte dei tuoi metodi fosse priva di effetti collaterali. Naturalmente non tutti i metodi possono essere di questo tipo (qualcuno dovrà pur modificarli, questi benedetti oggetti!), ma molti possono. Quindi:

1) se un metodo modifica un oggetto, chiediti se non abbia senso trasferirlo sull'oggetto stesso;
2) se un metodo fa più di una cosa insieme (ad esempio modifica i parametri e restituisce un valore), controlla che non sia il caso di spezzarlo in due metodi diversi;
3) dai a ciascun metodo un nome che chiarisce il suo scopo, come *incrementaNumero()* o *chiudiConnessioneAlDatabase()*.

NON CAMBI PROPRIO MAI

Ecco un esempio di oggetto il cui valore non può cambiare:

```
class NumeroImmutabile {
  private int num;
  Numero(int x) {num = x; }
  int getNumero() {return num; } }
```

Questa classe è quasi identica alla classe Numero, con un'importante differenza: il campo num è private. Quindi non è possibile accedervi direttamente:

```
NumeroImmutabile x = new NumeroImmutabile(10);
x.num++; // errore di compilazione
```

Il valore di num viene deciso nel costruttore, dopodiché nessuno può più modificarlo. Ma come è impossibile scrivere il valore del campo, così è impossibile leggerlo. Per questo motivo ho scritto il metodo *getNumero()*, che restituisce il valore del campo. Ora la proprietà *Numero* è una proprietà di sola lettura (se tutto questo ti confonde, rileggi l'articolo del mese scorso). Quindi il fatto che un oggetto

UGUALE, NON UGUALE

Il modo in cui Java gestisce gli oggetti può avere risultati sorprendenti. Un tipico errore dei principianti è usare l'operatore di confronto per confrontare i valori di due oggetti:

```
String s1 = "abc";
String s2 = "abc";
System.out.println(s1
  == s2); // stampa false!
```

L'operatore di confronto non confronta i valori dei due oggetti, ma i valori dei due reference. Questa operazione, che in apparenza è sensata, ha come risultato false perché i due reference puntano ad oggetti diversi.

sia o meno immutabile non è un problema di linguaggio, ma una questione di design. Non puoi semplicemente dichiarare un oggetto "immutabile" e contare sul fatto che Java impedisca di modificarlo. A dire il vero esiste un meccanismo del linguaggio che aiuta a rendere immutabili gli oggetti, ed è la parola chiave *final:*

```
class AltroNumeroImmutabile {
  final int num;
  final int altroNum = 7;
  AltroNumeroImmutabile(int x) {num = x; } }
```

Un campo *final* non può mai essere modificato. Il suo valore può essere assegnato in due posti: o nel punto di definizione del campo (come nel caso di *altroNum,* che varrà sempre e solo 7) o nel costruttore dell'oggetto (come nel caso di *num,* il cui valore può essere assegnato una sola volta). In tutti gli altri casi il compilatore ti impedirà di assegnare un valore ad un campo *final*:

```
AltroNumeroImmutabile x =
                    new AltroNumeroImmutabile(10);
System.out.println(x.num);  // stampa 10
x.num++; // errore di compilazione
```

Questo è un modo semplice per rendere un campo "di sola lettura". Personalmente preferisco dichiarare private i campi e dare l'accesso in lettura attraverso un metodo, come nel caso precedente. Questa forma richiedere più righe di codice, ma rende esplicito il fatto che il campo non può essere modificato. Di solito si usano le due tecniche insieme, per avere il massimo di chiarezza nel codice e un po' di aiuto da parte del compilatore in caso di errore:

```
class TerzoNumeroImmutabile {
  // questo campo e' final, quindi immutabile.
  // deve essere inizializzato nel costruttore.
  private final int num;
  Numero(int x) {num = x; }
  int getNumero() {return num;} }
```

Gli oggetti immutabili ci garantiscono una certa pace d'animo: possiamo passarli ai metodi e assegnarli in giro senza paura che qualcuno ne modifichi il valore causando un bug in qualche altra parte del programma. Infatti sono ampiamente usati nelle classi standard del linguaggio Java. Ad esempio, la classe *String*, che è immutabile. Se concateni due stringhe con l'operatore '+' il risultato è una terza stringa, e le due stringhe originali restano immutate. Ovviamente questa continua creazione e copia di oggetti significa che le operazioni sulle stringhe in Java possono diventare lente, ma questo di solito non è un problema (se proprio ti serve una stringa modificabile per questioni di prestazioni, Java ha una classe *StringBuffer* che serve proprio a questo). Stai attento a non cadere nel classico tranello che la parola chiave final tende ai principianti...

LA TRAPPOLA FINALE

Questa classe non funziona:

```
class NumeroComplesso {
  final Numero r;
  final Numero i;
  NumeroComplesso(int parteReale, int
                              parteImmaginaria) {
    r = new Numero(parteReale);
    i = new Numero(parteImmaginaria); }
  public void stampa() {
    System.out.println("(" + r.num + ", " + i.num + ")");}
}
```

Il *NumeroComplesso* è composto da una parte reale e da una immaginaria, che sono oggetti di classe *Numero*. Visto che tutti i campi di *NumeroComplesso* sono *final,* potremmo pensare che l'oggetto sia immutabile. Sbaglieremmo:

```
class Trappola {
  public static void main(String[] args) {
    NumeroComplesso x = new NumeroComplesso(3, 5);
    x.stampa();
    x.r.num = 100;  // !!!
    x.stampa(); }
}
```

Ecco l'output di *Trappola*:

```
(3,5)
(100, 5)
```

La parola chiave final, applicata ad un oggetto, non garantisce che l'oggetto sia immutabile. Quello che garantisce è che il reference è immutabile, cioé che farà sempre riferimento allo stesso oggetto:

```
NumeroComplesso x = new NumeroComplesso(3, 5);
x.r = new Numero(100);  // errore di compilazione
```

Ma nel nostro caso abbiamo potuto tranquillamente entrare nell'oggetto x.r (che non è immutabile) e cambiarne il valore: *x.r.num = 100; // OK.*
Quindi perché un oggetto sia immutabile i suoi campi pubblici devono essere a loro volta immutabili: l'immutabilità non è un problema tecnico, ma un problema di design. La parola chiave *final* può aiutarti, ma alla fine quello che conta è la tua attenzione quando progetti le classi.

Paolo Perrotta

NOTA

VALORI E IDENTITÀ

Se copiamo il reference di una stringa in un altro reference, il confronto risulta vero:

```
String s3 = s1;
System.out.println(s1 ==
        s3);  // stampa true
```

In questo caso i due reference puntano allo stesso oggetto, quindi sono uguali. Ricorda sempre che l'operatore di confronto applicato agli oggetti significa: "dimmi se questi due oggetti sono in realtà lo stesso oggetto", non: "dimmi se questi due oggetti hanno lo stesso valore".

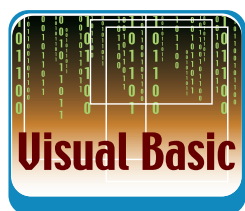
CONFRONTI FRA STRINGHE

Come fare a confrontare due stringhe? La classe String ha un metodo equals() che prende un'altra stringa, e restituisce true se e solo se la stringa passata come parametro ha lo stesso valore di quella sulla quale hai chiamato il metodo:

```
System.out.println(s1.eq
    uals(s2)); // stampa true
```

Realizziamo un client per la gestione dei fax

Fax Service SDK senza segreti

In questo articolo descriveremo come implementare un Fax Client utilizzando gli elementi che permettono di interagire pienamente con i Servizi Fax messi a disposizione da Windows.

TIFF

Il formato Tiff (Tag Image File Format) ha la capacità di contenere insieme alle immagini il testo riconoscibile dalla funzione di riconoscimento ottico dei caratteri (OCR - Optical Character Recognition). Il formato TIFF ha sei derivati: No-compressin, Huffman, Pack Bits, LZW, Fax Group 3 e Fax Group 4. Il formato Tiff accettato dal fax server è non compresso, con ampiezza 1728 pixels, con X-risoluzione <= 204dpi, con Y-risoluzione <= 200dpi e con Y-risoluzione uguale per tutte le pagine.

Nel precedente appuntamento abbiamo descritto varie tecniche per inviare e ricevere fax, e-mail e file di dati. Ci siamo occupati del controllo *MsComm*, dei controlli MAPI (*MAPISession* e *MAPIMessages*) e di come interagire con Outlook per amministrare la rubrica contatti, i fax e le e-mail. Come esempio, abbiamo introdotto un progetto Visual Basic che consente di inviare e ricevere fax attraverso la piattaforma Visual Basic-Outlook. In questo appuntamento, invece, descriveremo come realizzare un *Client Fax* con le *Fax Service Extended COM API,* le estensioni COM che consentono di interagire con i servizi fax di Windows (anche su rete locale). Oltre ad introdurre un'applicazione *Fax Client* descriveremo i tool presenti nei sistemi operativi Microsoft, che permettono di gestire documenti in formato *TIFF* (il formato dei documenti fax) e *COV* (il formato della *cover page* – frontespizio - del fax) ed inviare fax. In particolare vedremo la *Console Servizio Fax,* cioè la console che consente, all'utente, di amministrare i fax, il *MODI (Microsoft Office Document Imaging)* e l'*Editor di Frontespizio.* Il *MODI* fa parte degli strumenti di Office mentre l'*Editor di Frontespizio* è fornito insieme alla *Console Servizio Fax.* Precisiamo che gli esempi che presenteremo sono stati implementati e testati con il sistema operativo Windows XP Professional.

FAX SERVICE SDK

In Windows XP, la *Console Servizio Fax* è un servizio fax completo che permette all'utente di creare, inviare, ricevere, monitorare ed archiviare fax. Attraverso le funzionalità della *Console Servizio Fax* è anche possibile configurare la stampante per i fax, inviare dei frontespizi predefiniti o personalizzati ecc. Grazie a questo strumento ed al Fax Service SDK, sostanzialmente, nei sistemi operativi Windows 2000/XP /2003, possiamo inviare fax in almeno quattro modi, e cioè:

1. Stampando da un'applicazione (per esempio Word). In questo caso l'applicazione e i drive del Fax Service si preoccuperanno di convertire i documenti in formato TIFF (estensione *.tif*).

2. Inviando solo una cover-page, questo si può fare attraverso il Wizard della Console (selezionabile da *Start/Programmi/comunicazione/fax/invia fax*), anche in questo caso c'è una pre-conversione del documento in formato TIFF.

3. Utilizzando Outlook 2000/2002/2003 come abbiamo fatto nel precedente appuntamento.

4. Attraverso un fax client implementato utilizzando le fax COM API. In questo caso le funzionalità di basso livello si preoccuperanno della conversione dei documenti in formato TIFF.

Il *Microsoft Fax Service Software Development Kit* (SDK) può essere utilizzato per sviluppare applicazioni per i sistemi Windows 2000/XP/ 2003, ma le applicazioni fax client possono essere eseguite anche su Windows NT/95/98 in un ambiente Win32.

FAX SERVICE EXTENDED COM API

Le *Fax Service Extended COM API* fanno parte del *Fax Service SDK* e permettono di incorporare le funzionalità fax per l'utente nelle ap-

plicazioni, in sostanza consentono a Visual Basic di interagire con il *Fax Service Extended COM Object Model* (FSECOM) che appunto è alla base dei servizi fax. Gli oggetti e le interfacce del *FSECOM* sono divisi nei seguenti gruppi: *Root Object* che include l'oggetto *FaxServer* e le funzioni per connettersi e disconnettersi ad un fax service attivo; *Messaging Objects* che include gli oggetti usati per creare e gestire documenti *(FaxDocument)*, messaggi, informazioni sui destinatari dei fax *(FaxRecipient)* ecc; *Configuration Objects* che include gli oggetti per configurare il servizio fax; *Notification Objects* che include gli oggetti per la notificazione di eventi (di *callbacks*).

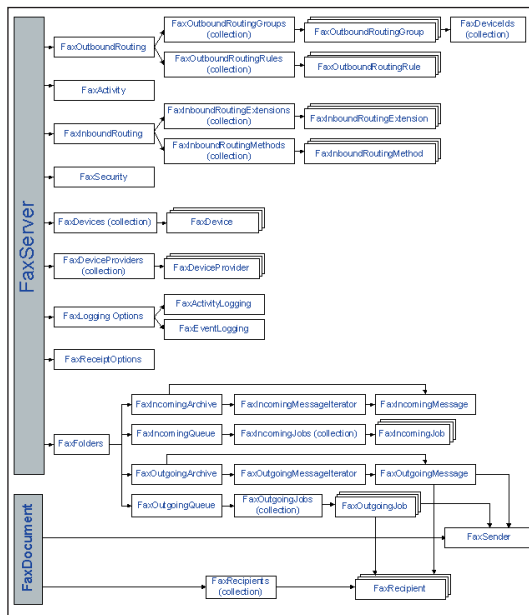



Fig. 1:* *Il Fax Service Extended COM Object Model.

Come avete potuto intuire da questa breve descrizione, il modello mette a disposizione diversi oggetti e quindi diverse funzionalità. Nel corso dell'articolo descriveremo soltanto gli elementi indispensabili per l'implementazione di un semplice Fax Client (lasciandovi il compito di studiare gli altri oggetti del modello e quindi di ampliare le funzionalità dell'applicazione). Ora descriviamo brevemente gli oggetti *FaxServer* e *FaxDocument* che sono fondamentali per il nostro esempio.

FaxServer E FaxDocument

L'oggetto *FaxServer* è usato per creare e gestire una connessione con un fax server. Esso è l'oggetto radice del modello ad oggetti per la gestione dei servizi fax; dopo aver creato una sua istanza è possibile accedere ad altri oggetti, però non è necessaria per accedere ad un oggetto *FaxDocument*. Per creare un oggetto *FaxServer* (o un *FaxDocument*) si possono usare i due modi seguenti:

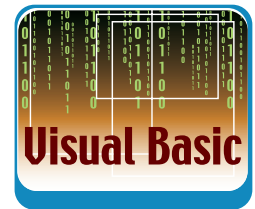
```
Set FaxServer = CreateObject("FaxComEx.FaxServer")
Dim objFaxDocument As New
                        FAXCOMEXLib.FaxDocument
```

Per creare una connessione si deve usare il metodo *Connect* con il nome del server. Se il server è locale non deve essere specificato.

```
objFaxServer.Connect "nomecomputer"
`quando il server è in rete
objFaxServer.Connect ""
`quando il server è locale
```

L'oggetto *FaxDocument* serve per comporre un documento da inviare al fax server che provvederà ad elaborarlo. Un *FaxDocument* contiene gli oggetti *FaxSender* e *FaxRecipients*.
Con *FaxSender* si specificano le informazioni sul mittente, mentre con i *FaxRecipient* (contenuti nella collezione *FaxRecipients*) si specificano informazioni sui destinatari del fax.
Con *FaxDocument* inoltre si può specificare la Cover Page del fax (che costituirà la prima pagina del fax), il soggetto (proprietà *subject*), le note della cover page (proprietà *note*), si può pianificare quando inviare il fax (proprietà *scheduletime*), definire una ricevuta di ritorno per il fax (proprietà *AttachFaxToReceipt*, *ReceiptAddress* e *ReceiptType*), ecc. Inoltre *FaxDocument* fornisce i metodi che consentono di sottoporre il fax al fax server (*Submit* e *ConnectedSubmit*).
Dunque per inviare un fax, devono essere impostate le seguenti azioni: creare un'istanza dell'oggetto *FaxServer*, stabilire una connessione con il fax server attivo, creare un'istanza dell'oggetto *FaxDocument* (specificando il corpo del fax, con *Body* e *Recipient*, i dati del ricevente) e sottomettere il tutto al fax server.

IL FAX CLIENT

Il form del progetto di esempio è analogo a quello visto nel precedente appuntamento ed è mostrato in Fig. 2. Rispetto al precedente esempio, abbiamo introdotto gli elementi che gestiscono un timer (che serve per pianificare l'invio dei fax) ed inserito un riferimento alla libreria *Microsoft Fax Service Extended COM Type Library (fxscomex.dll)*. Come abbiamo

MODI

***Microsoft Office Document Imaging* (*MODI*) consente di gestire un documento digitalizzato come qualsiasi altro documento di Microsoft Office. Esso è utile per fare l'anteprima dei fax presenti nelle cartelle della *Console Servizio Fax*, fare il riconoscimento ottico dei caratteri da immagini digitalizzate, esportare o salvare il contenuto dell'anteprima dei fax, modificare i fax ecc. Inoltre facciamo notare che in office per salvare un documento in formato *Tiff* basta stampare il documento utilizzando il driver di stampa *Microsoft Office Document Image Writer*, un componente dell'applicazione *Microsoft Office Document Imaging* installato con Office.**

EDITOR FRONTESPIZIO

L'editor di Frontespizio di Windows XP (selezionabile da Start/.../fax/editor di frontespizio) consente di disegnare lo schema del frontespizio e predisporre i campi per vari tipi di dati (mittente, destinatario e informazioni sul fax). Inoltre permette d'inserire delle immagini attraverso gli appunti di Windows (cioè attraverso il copia - incolla).

Fig. 2: ***Il form del Fax Client.***

visto nel precedente appuntamento, attraverso gli oggetti *MAPI* interagiamo con la rubrica di Outlook per ottenere i dati dei destinatari. In questo caso, però, il numero di fax deve essere fornito senza i caratteri alfanumerici aggiunti da Outlook (che invece sono necessari, quando si invia un fax con Outlook). Per questo è necessario definire una procedura di conversione che elimini i caratteri spuri. In base a quello descritto sopra e nel precedente articolo per inviare un fax possiamo definire la seguente procedura.

```
Private Sub inviofax_Click()
 Dim objFaxServer As New FAXCOMEXLib.FaxServer
 Dim objFaxDocument As New
                                    FAXCOMEXLib.FaxDocument
 Dim JobID As Variant
 objFaxServer.Connect ""
 If Not controlladati Then
  MsgBox "Manca qualche dato", vbExclamation, "Errore"
  Exit Sub
 End If
 objFaxDocument.Body = Me.txtallegato
 'nome del documento
 objFaxDocument.DocumentName = "fax max"
 'imposta la priorità
 objFaxDocument.Priority = fptHIGH
 'inserisce numero fax e nome destinatario
 objFaxDocument.Recipients.Add
eliminacaratteri(txtNumFax) , txtdest
 'imposta la cover page
 objFaxDocument.CoverPageType = fcptLOCAL
 objFaxDocument.CoverPage =
                              "...specificare path\confiden.cov"
 'inserisce le note per la cover page
 objFaxDocument.Note = Me.txtbody
 objFaxDocument.Subject = Me.txtoggetto
 JobID = objFaxDocument.ConnectedSubmit(objFaxServer)
 Set objFaxDocument = Nothing
 Set objFaxServer = Nothing
End Sub
```

```
Private Function eliminacaratteri(nume As String)
                                                    As String
 If InStr(nume, "@") Then
   pos = InStr(1, nume, "@")
   eliminacaratteri = Mid(nume, pos + 1)
  Else
  eliminacaratteri = nume
 End If
End Function
```

Notate che il metodo *ConnectedSubmit* restituisce un array di fax *JobId*, uno per ogni ricevente (*recipient*). Il *JobId* può essere utilizzato per determinare lo stato del fax attraverso l'oggetto *FaxJobStatus* (si controllino i valori della costante *FAX_JOB_STATUS_ENUM*). Attenzione, però, il metodo *ConnectedSubmit* per funzionare correttamente richiede che il *FaxDocument* abbia almeno un recipient e la cover page o il fax body. Per inserire i dati del mittente, sulla cover page, possiamo utilizzare l'oggetto *Sender* contenuto nel *FaxDocument*.

```
objFaxDocument.Sender.Title = "Dott."
objFaxDocument.Sender.Name = "Nicola"
objFaxDocument.Sender.City = "Lecce"
objFaxDocument.Sender.FaxNumber = "12345678"
objFaxDocument.Sender.OfficeLocation = "Milano"
...
```

Naturalmente queste istruzioni devono essere aggiunte alla procedura predente.

FRONTESPIZIO PERSONALIZZATO

Come frontespizio dei fax possiamo usare uno di quelli predefiniti (generico, urgente, confiden, ecc.) oppure costruirne uno con l'*Editor di Frontespizio* di Windows.
Il frontespizio non è altro che un file, con estensione *"cov"*, che al suo interno contiene immagini e campi, questi ultimi possono essere impostati, a run-time, dall'utente o da un Fax Client. Con l'editor di frontespizio i

Fig. 3: ***Il Frontespizio ioProgrammo creato con l'Editor di Frontespizio.***

campi vengono inseriti attraverso il menu *inserisci*. In particolare è possibile inserire i campi che conterranno informazioni sul mittente, sul destinatario e messaggi *(oggetto, data, ora, numero pagine)*. Per esempio noi abbiamo creato il frontespizio di ioProgrammo con varie informazioni, come potete controllare in Fig. 3.
Nell'esempio descritto sopra invece abbiamo usato la cover page *"confiden.cov"*.

FAX CON RICEVUTA DI RITORNO

Come accennato, attraverso le proprietà di *FaxDocument* è possibile impostare la ricevuta di consegna del fax. Questa può essere una email o un *msgbox*. Per esempio, con le seguenti istruzioni viene specificato che si vuole una ricevuta attraverso un *MsgBox*.

```
objFaxDocument.AttachFaxToReceipt = True
objFaxDocument.ReceiptAddress = "nomecomputer"
objFaxDocument.ReceiptType = frtMSGBOX
```

Con le istruzioni precedenti s'impostano le seguenti azioni: si attacca una ricevuta al fax, si specifica il nome del computer che avrà la ricevuta, si specifica il tipo di ricevuta.
In conclusione vediamo le istruzioni per pianificare l'invio dei fax. Per tale scopo sul form abbiamo previsto un Frame (nominato *Frametime*), un Timer, un DtPicker (nominato *DtFax*), un CheckBox (nominato *ImpTime*) e una label (nominata *LabTime*). In particolare quando il CheckBox è selezionato la label mostra l'orario corrente e il DtPicker è impostato sul valore della label più 10 minuti. Di seguito riportiamo il codice da prevedere per gestire l'orario e pianificare l'invio del fax.

```
Private Sub Timer1_Timer()
 'bisogna impostare la proprietà Timer1.Interval
 LabTime = Time
End Sub

Private Sub Imptime_Click()
 If Imptime.Value = 1 Then
  Frametime.Enabled = True
  Timer1.Enabled = True
  LabTime = Time
  DTfax.Hour = Hour(Time)
  If Minute(Time) + 10 >= 60 Then
   DTfax.Hour = Hour(Time) + 1
   DTfax.Minute = 10 - (60 - Minute(Time))
  Else
   DTfax.Minute = Minute(Time) + 10
```

```
  End If
  DTfax.Second = Second(Time)
 Else
  Timer1.Enabled = False
  Frametime.Enabled = False
 End If
End Sub
```

Infine definiamo il codice d'aggiungere alla *inviofax_Click*:

```
If Imptime = 1 Then
 objFaxDocument.ScheduleType = fstSPECIFIC_TIME
 objFaxDocument.ScheduleTime = DTfax.Value
End If
```

Con il codice precedente specifichiamo il tipo di pianificazione (cioè che il fax deve essere spedito all'ora specificata e rispettando la priorità impostata) e l'ora d'invio.

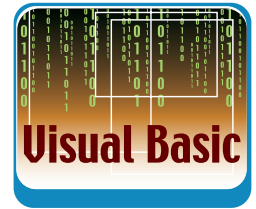
RICEVERE FAX

La ricezione dei fax può essere gestita attraverso la Console Servizi Fax oppure interrogando la collezione di oggetti *FaxIncomingJobs* (che è una collezione di oggetti *FaxIncomingJob*) cioè i fax job che si trovano nella lista di fax in ingresso; oppure l'oggetto *FaxIncomingArchive* ecc.
Per esempio con le seguenti istruzioni si stabilisce se ci sono fax nella coda dei fax in ingresso:

```
 Dim objFaxServer As New FAXCOMEXLib.FaxServer
 Dim collFaxIncomingJobs As FaxIncomingJobs
 ...
 objFaxServer.Connect ""
 Set collFaxIncomingJobs =
objFaxServer.Folders.IncomingQueue.GetJobs
 MsgBox "ci sono " & collFaxIncomingJobs.Count & "
jobs nella coda d'ingresso"
 ...
```

CONCLUSIONI

In questo appuntamento abbiamo introdotto gli strumenti per la gestione dei fax, in particolare abbiamo presentato un'applicazione che utilizza alcuni elementi delle *Fax Service Extended COM API*. Queste fanno parte del Microsoft Fax Service SDK che contiene anche i seguenti elementi: *Fax Service Provider API, Fax Routing Extension API, Fax Extension Configuration* e *MMC Snap-in Node Types*.

Massimo Autiero

Cust tag e gestione degli errori

Un Portale in Java parte terza

Concludiamo la realizzazione del portale, imparando a definire i custom tag. Molta attenzione sarà dedicata alla gestione degli errori, indispensabile nei migliori progetti professionali.

Negli articoli precedenti abbiamo creato le basi per la realizzazione di un portale verticale basato su architettura Java 2 Enterprise Edition. Nel primo articolo ci siamo concentrati sulle componenti di analisi funzionale e di disegno architetturale, mentre nel secondo abbiamo iniziato a realizzare i vari moduli dell'applicazione. In questa ultima puntata chiuderemo il cerchio, completando tutte le componenti applicative che ancora mancano all'appello. Avremo portato a casa l'obiettivo che ci eravamo prefissi, cioè la realizzazione di un portale completo basato su architettura Java 2 Enterprise Edition e che utilizza il framework *Struts* come semilavorato, che ci consente di avere già implementati tutta una serie di servizi comuni ed in particolare la completa gestione del pattern MVC.

LE COMPONENTI DELLA PAGINA

Negli articoli precedenti abbiamo visto il funzionamento del controller MVC che ci consente, attraverso un meccanismo di template, di definire all'interno di ogni action quale sarà il componente di visualizzazione mostrato al centro della pagina del Portale e che rappresenta il vero e proprio contenuto che vogliamo erogare all'utenza. Le altre componenti della pagina sono gestite nativamente dal template di struttura che è implementato attraverso la pagina *home_template.jsp*. Si tratta in particolare di una testata nella parte superiore, di un menù nella zona di sinistra, di una spalla contenente un form di login nella zona di destra. Queste zone della pagina sono posizionate attraverso il template di struttura e sono implementate da una serie di pagine JSP, si tratta in particolare delle pagine: *testata.jsp*, *menu.jsp* e *login.jsp*. Il menù ed il box di login, invece, seppur realizzati in maniera semplice, sono costituiti in realtà da due Custom Tag e di conseguenza implementati attraverso classi Java specifiche. Il motivo di questa scelta risiede nel fatto che sia il menù sia il box di login sono componenti ad alta personalizzazione i cui contenuti possono variare a seconda del contesto. Si pensi per esempio ad una situazione in cui le voci di menù, cioè le funzioni cui l'utente può accedere, siano dipendenti da un determinato profilo dell'utente, è chiaro che un utente amministratore potrebbe avere un menù diverso da un semplice utente occasionale o da un utente registrato ad una serie di servizi del Portale. Se non si fosse spostata la logica di produzione del menù nelle classi Java saremmo stati obbligati a contenerla all'interno della pagine JSP stesse, contravvenendo ad una delle regole di buona programmazione che dice, appunto, che la separazione tra i livelli deve essere il più possibile marcata. Nel nostro caso quindi la pagina JSP, che per sua natura si occupa di generare il rendering della pagina, non deve contenere logica applicativa, ma quest'ultima deve essere posizionata nella componente di *Model* dell'architettura.

***Fig. 1:** La Home Page del Portale.*

LA LIBRERIA DI CUSTOM TAG

Quello che ci serve quindi è una libreria di tag personalizzati che si occupino di generare il codice html delle porzioni di pagina che ci interessano, in particolare costruiremo un tag specifico per la generazione del menù ed uno per la generazione del box di login e questi saranno descritti all'interno di un file di definizione, si tratta in particolare del file *WEB-INF/fish.tld*, vediamone il contenuto:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems,
  Inc.//DTD JSP Tag Library 1.1//EN" "http://java.sun.com
            /j2ee/dtds/web-jsptaglibrary_1_1.dtd">
<taglib>
  <tlibversion>1.0</tlibversion>
  <jspversion>1.1</jspversion>
  <shortname>fishlib</shortname>
  <tag>
```

```
      <name>menugenerator</name>
      <tagclass>com.fish.tags.MenuGenerator</tagclass>
      <bodycontent>empty</bodycontent>
      <info>Tag che genera il menù dell'utente</info>
   </tag>
   <tag>
      <name>logingenerator</name>
      <tagclass>com.fish.tags.LoginGenerator</tagclass>
      <bodycontent>empty</bodycontent>
      <info>Tag che genera il box di loign</info>
   </tag>
</taglib>
```

Come si può vedere, questo file contiene la definizione della libreria di tag che stiamo costruendo e contiene sia informazioni legate alla libreria stessa, come ad esempio il nome, sia le definizioni dei due tag. Analizzandone il contenuto si può vedere che i due tag che stiamo definendo si chiamano menugenerator (implementato dalla classe *com.fish.tags .MenuGenerator*) e *logingenerator* (implementato dalla classe *com.fish.tags.LoginGenerator*). Una volta generato il file di definizione della libreria di tag è essenziale referenziarlo all'interno del file *web.xml* che descrive molti degli aspetti dell'applicazione web. In particolare in questo file è necessario inserire il seguente frammento XML:

```
<taglib>
    <taglib-uri>/tags/fish</taglib-uri>
    <taglib-location>/WEB-INF/fish.tld</taglib-location>
</taglib>
```

In questo modo si rende disponibile all'applicazione una nuova libreria di tag referenziabile all'url */tags/fish* e definita fisicamente nel file */WEB-INF/fish.tld.*

IMPLEMENTAZIONE DEI CUSTOM TAG

Adesso che abbiamo descritto la libreria di tag è giunto il momento di concretizzare il lavoro scrivendo il codice delle le classi, in Fig. 2 possiamo vedere il semplice contenuto del package *com.fish.tags.*

Class Diagram del package com.fish.tags

Le due classi sono praticamente identiche. Vediamo, a titolo di esempio, il codice della classe che si occupa della generazione del menù:

```
package com.fish.tags;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.TagSupport;
public class LoginGenerator extends TagSupport {
    public int doStartTag() {
        JspWriter out = pageContext.getOut();
        try {
            String content = "";
            content += "codice html che costituisce il tag";
            out.println(content); }
        catch (Exception ex) {
            try {
            out.println("<META HTTP-EQUIV=\"refresh\"
              content=\"0;URL=pages/globalerror.jsp\"> "); }
            catch ( Exception e){
                e.printStackTrace(); } }
        return SKIP_BODY; }
}
```

In questo frammento di codice si vede chiaramente che una classe che implementa un tag deve estendere la classe *javax.servlet.jsp.tagext.TagSupport* e di questa deve (almeno) effettuare l'overloading del metodo *doStartTag()*. In particolare in questo metodo viene prodotto il codice html che deve essere spedito al client quando il tag viene aperto nella pagina JSP, nel nostro caso utilizzeremo tag senza contenuto, cioè nella forma contratta *<tag />* e senza parametri, pertanto l'implementazione di questo metodo è più che sufficiente per le nostre esigenze. Il codice html prodotto viene poi rispedito sul client attraverso l'istruzione:

```
out.println(content);
```

La gestione degli errori sarà analizzata nel dettaglio successivamente. A questo punto appare chiaro il motivo per cui si è scelto di utilizzare un tag per la produzione di queste due delicate componenti applicative, il menù e il box di login. Con questa struttura è infatti possibile delegare ad altri oggetti di business la produzione di frammenti di codice legati, ad esempio, alla profilatura dell'utente che in quel momento sta visitando il portale, pertanto in un tag è possibile eseguire tranquillamente qualsiasi operazione di business (cioè legata alla componente di Model dell'applicazione) senza dover mettere nemmeno una riga di codice applicativo all'interno delle pagine JSP che utilizzano il tag stesso. In particolari contesti è possibile, al limite, passare al tag dei parametri direttamente dal codice della pagina JSP rileggerli dalla classe che implementa il tag per avere informazioni aggiuntive legate allo specifico contesto di esecuzione. Nel nostro caso però la situazione è più semplice e la pagina JSP che si occupa di fare da contenitore per il tag contiene esclusivamente le informazioni legate al tag stesso, vediamo ad esempio la pagina *menu.jsp* che si occupa di contenere il tag del menù, si tratta di una pagina JSP di ben due righe:

```
<%@ taglib uri="/tags/fish" prefix="fishlib" %>
<fishlib:menugenerator />
```

La ragione di tanta semplicità sta nel fatto che tutta

J2EE

La complessa architettura di Java è divise in tre piattaforme distinte. Java 2 Standard Edition (J2SE) definisce i costrutti del linguaggio, le librerie di utilizzo generale ed i suoi servizi di base. Java 2 Micro Edition (J2ME) consente l produzione di applicazioni per dispositivi wireless mobili e per sistemi embedded. Java 2 Enterprirse Edition (J2EE) è orientata allo sviluppo di applicazioni distribuite su sistemi Internet/Intranet e consente la produzione di componenti server di elevata complessità attraverso le tecnologie JSP, Servlet, EJB e Web Services. La piattaforma si completa attraverso l'adozione di consolidati standard di integrazione come JMS, JTS, JCA, JNDI.

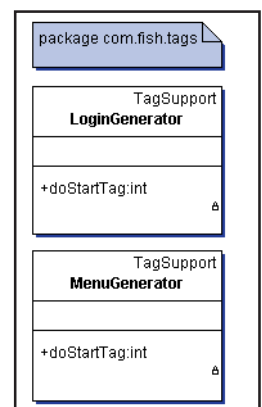



***Fig. 2:** La struttura del package tags.*

la logica dedicata alla generazione del menù è contenuta nel tag stesso: la pagina JSP non deve fare altro che istanziare la libreria ed usare il tag più opportuno. A questo punto abbiamo tutte le componenti che ci servono per costruire il nostro Portale. Il portale sarebbe praticamente finito, se non fosse che, come si sa, l'imprevisto è sempre dietro l'angolo, pertanto è necessario che il nostro portale si sappia comportare in maniera educata di fronte a situazioni di errore e mostri un messaggio controllato all'utenza.

LA GESTIONE DEGLI ERRORI

Gli errori che possono occorrere in un'applicazione di questo tipo sono generalmente di due tipi: errori locali ed errori globali. Gli errori locali coinvolgono generalmente una singola funzione o un gruppo di funzioni del portale, ma consentono al navigatore di continuare ad utilizzare le altre. Si pensi ad esempio ad una situazione in cui non funziona uno dei canali tematici del portale per un errore su qualche tabella del database. Quel singolo canale è certamente bloccato, ma può essere che gli altri funzionino, pertanto è necessario garantire all'utente la possibilità di utilizzare i servizi che restano. Quello che faremo per gestire questo errore sarà quindi realizzare una pagina di cortesia che comunichi all'utente l'impossibilità di utilizzare quello specifico servizio, lasciando però ampia liberta per la navigazione tra le altre funzionalità. Gli errori globali coinvolgono generalmente una componente strutturale del portale, di conseguenza non è possibile consentire all'utente l'utilizzo di nessun servizio.
Quello che faremo in questo caso sarà mostrare un messaggio di cortesia che non consenta la navigazione tra i servizi, cosa che comunque sarebbe impossibile.

STRUMENTI DI PROGETTAZIONE

La progettazione di questo portale ed il design dei diagrammi UML sono stati realizzati utilizzando il tool Borland Together Controlcenter 6.01, uno strumento commerciale completo per la realizzazione di applicazioni Java, dalla progettazione alla realizzazione. E' possibile scaricare una versione di valutazione di Together, oltre alla documentazione, all'indirizzo: http://www.borland.com/together/controlcenter/

GLI ERRORI LOCALI

Per implementare la gestione degli errori locali ci affideremo al framework Struts: all'interno delle classi che implementano le *Action*, è possibile inviare messaggi diversi a seconda dei risultati delle varie elaborazioni. Proviamo a guardare una parte del codice del metodo execute classe *HomeAction:*

```
public ActionForward execute(
    ActionMapping mapping,
    ActionForm form,
    HttpServletRequest request,
    HttpServletResponse response)
    throws Exception {
    try {
    // produzione dei contenuti per la Home Page
    return mapping.findForward("success");}
    catch (RuntimeException e) {
        this.getErrorBean().setError("Impossibile
                caricare la Home Page",(SessionBean)
    request.getSession().getAttribute("SessionBean"));
    request.getSession().setAttribute("errorBean",
                                this.getErrorBean());
        return mapping.findForward("failure"); }
  }
```

Come vedete, il codice della classe è blindato all'interno di un blocco *try/catch*, pertanto se le cose vanno bene e non si generano e non vengono sollevate eccezioni, l'esecuzione del metodo termina attraverso l'istruzione *return mapping.findForward ("success*"); che spedisce alla componente di controllo il messaggio che conferma la corretta esecuzione. Se però viene sollevata una qualsiasi eccezione, allora il controllo passa al blocco *catch* che esegue le seguenti operazioni:

- recupera un particolare bean: *ErrorBean,* e ne esegue il metodo *setError* con il quale si vuole trasmettere il messaggio di errore;
- inserisce il bean in sessione;
- restituisce un messaggio divero dal precedente, in paricolare il messaggio "failure".

Questo messaggio, se ricordate, è stato censito a livello di file di configurazione del framework ed in particolare era stato creato un global-forward specifico attraverso il frammento XML:

```
<forward name="failure" path="/error.do"/>
```

Questo global-forward non fa altro che redirigere l'attenzione dell'applicazione verso un'altra action, quella gestita dal path */error* che è configurato attraverso il seguente frammento XML:

```
<action  path="/error"
      scope="request"
      type="com.fish.actions.ErrorAction"
      validate="false">
   <forward name="success"
path="/pages/home.jsp"/>
</action>
```

Come si può vedere, la gestione dell'errore passa quindi alle attenzioni della classe *com.fish.actions. ErrorAction* e se questa ritornerà il messaggio "success", cioè avrà fatto il suo lavoro correttamente, allora il rendering dell'applicazione sarà affidato alla JSP che si occupa di presentare il template e di riempirlo. Il risultato quindi sarà qualcosa di molto simile al portale standard, ma con un contenuto diverso nella zona centrale. Per capire quale sarà que-

sto contenuto è necessario vedere il codice della classe *ErrorAction:*

```
package com.fish.actions;
import javax.servlet.http.HttpServletRequest;
...
public class ErrorAction extends com.fish.actions.BaseAction
{       public ActionForward execute(
        ActionMapping mapping,
        ActionForm form,
        HttpServletRequest request,
        HttpServletResponse response)
        throws Exception
    { try {
         this.getNavBean().setContent("error");
         request.getSession().setAttribute(
         "navigationBean",
         this.getNavBean());
         return mapping.findForward("success");
         } catch (RuntimeException e) {
             return mapping.findForward("globalerror");}
} }
```

Questo codice è molto semplice e non fa altro che inserire il valore "error" all'interno del bean di navigazione in modo che il rendering della parte centrale del portale sia affidato alla pagina *error.jsp*. Se qualcosa non funziona a livello di controllo degli errori locali, allora possiamo passare il controllo al gestore degli errori globali in modo da alzare il livello di errore e dare comunque un messaggio all'utente. Ecco il codice sorgente della pagina *error.jsp*:

```
<%@ page import="java.util.*" %>
<%@ page import="javax.servlet.*" %>
<%@ page import="com.fish.beans.ErrorBean" %>
<%@ taglib uri="/tags/struts-template" prefix=
                                        "template" %>
<%
   ErrorBean errorBean = (ErrorBean)session.getAttribute(
                                         "errorBean");
   String descError = errorBean.getDescError();
%>
<table border="0" cellpadding="5" align="center"
       cellspacing="0" style="border-collapse: collapse"
                                       width="90%">
   <tr>
       <td width="100%" class="bordo_giu" valign="top">
       <p  class="arial_13px_bold">
       <br/>Attenzione, si è verificato il seguente
                             errore applicativo: <br />
       <%= descError %><br/>
       </p>
       </td>
   </tr>
   <tr>
       <td>
       <br></br>
       </td>
   </tr>
</table>
```

Si tratta di una pagina che recupera dalla sessione il bean che contiene l'errore, da questo preleva il messaggio di errore da mostrare all'utente e lo inserisce nel codice html del portale.

GLI ERRORI GLOBALI

Abbiamo visto come gestire gli errori locali, ma cosa succede se a non funzionare è un componente fondamentale dell'applicazione come ad esempio il tag che genera il menù oppure lo stesso gestore degli errori locali? In questo caso la situazione è un po' più semplice da gestire in quanto, purtroppo, c'è ben poco da fare. Se analizziamo il codice della classe *ErrorAction* visto in precedenza è possibile notare che la gestione tradizionale dell'errore viene eseguita all'interno di un blocco try, se invece qualcosa non funziona si passa, come di consueto al blocco catch, il cui contenuto è una riga semplice e sibillina:

```
return mapping.findForward("globalerror");
```

il controllo passa cioè nelle mani del gestore degli errori globale attraverso un particolare globalforward censito nel file di configurazione di Struts:

```
<forward name="globalerror" path="/pages/
                            globalerror.jsp"  />
```

In questo caso, come si può vedere, il controllo passa semplicemente nelle mani di una pagina JSP che mostra un messaggio di cortesia all'utente in quanto non c'è nulla da fare per poter recuperare i controllo dell'applicazione.

CONCLUSIONI

Siamo arrivati quindi alla fine di questa serie di articoli dedicati alla realizzazione di un completo portale verticale basato su architettura J2EE ed appoggiato sul framework Struts. Abbiamo realizzato una completa infrastruttura di erogazione dei contenuti, un controller di secondo livello, un meccanismo di templating completo. Il tutto corredato da alcuni Custom Tag per la generazione di componenti particolarmente delicate della struttura della pagina e da una accurata gestione degli errori locali e globali. Questo è un buon semilavorato per iniziare a realizzare portali dotato di questi requisiti funzionali e tecnologici. Buon lavoro.

Massimo Canducci

IL FRAMEWORK STRUTS

Struts **è un applciation framework open source scritto utilizzando Servlet e JSP e quindi basato interamente sulla tecnologia Java 2 Enterprise Edition. Si tratta di un framework semplice e snello ma dalle caratteristiche molto interessanti, la sua architettura interna è completamente basata sul pattern *Model View Controller* che consente di separare tra loro le componenti applicative. È interamente configurabile dall'esterno attraverso l'utilizzo di file di configurazione XML che consentono di descrivere la struttura di navigazione del portale, le classi che devono essere eseguite in funzione delle scelte di navigazione dell'utente e le componenti *view* che si devono occupare del rendering dell'informazione. Struts è parte del progetto Jakarta ed è scaricabile dall'url:** http://jakarta.apache.org/struts/

Leggere un modello ed esportare il codice

Scrivere un generatore di codice in Java

parte seconda

In questo secondo appuntamento dedicato alla generazione del codice, mostreremo come progettare e sviluppare un generatore di codice flessibile e robusto su piattaforma Java.

Un generatore di codice è un programma che riceve in input un modello e produce in output codice sorgente che implementa tale modello. Il modello consiste in un insieme di metadati contenente informazioni in merito al codice che deve essere generato. Esso può essere uno schema UML, un file proprietario o perfino altro codice sorgente. Il formato è generalmente XML, testo, file CSV o sorgenti di altra natura quali directory, database e repository. Partendo dalle informazioni contenute nel modello, il generatore di codice crea codice sorgente per un dato linguaggio di programmazione (C, C++, Java, C#, VB, ecc...) oppure altri tipi di documenti tipo descrittori, file di configurazione, codice SQL o documentazione. In questo articolo tracceremo le linee guida per progettare ed implementare un generatore di codice generico, flessibile ed efficiente usando il linguaggio Java. Sebbene i concetti esposti saranno del tutto generici, gli esempi mostreranno come generare codice Java a partire dalla rappresentazione UML di un class diagram. Per motivi di spazio, alcuni dei listati che incontreremo saranno implementati solo in parte. Sul CD allegato alla rivista, tuttavia, è presente il codice completo del generatore oggetto della nostra attenzione.

Fig. 1: Architettura del generatore di codice.

ARCHITETTURA DEL GENERATORE

Il generatore di codice che andremo a sviluppare seguirà parzialmente la filosofia *MDA* introdotta lo scorso mese. Diciamo parzialmente poiché esso, essendo particolarmente flessibile, potrà essere usato anche per approcci *code-driven* e *model-drive* custom. Il generatore sarà composto da tre moduli differenti:

Importer - legge il modello in input (non necessariamente UML) e lo converte in un formato platform -independent basato su un modello ad oggetti. L'importer può essere visto come una sorta di parser DOM ma, come vedremo, esso potrà essere implementato per accettare qualsiasi tipo di input si desideri.

Internal Object Model *(IOM)* - è il formato intermedio che può essere visto come la parte *core* del generatore di codice. Lo *IOM* contiene un insieme di classi che rendono semplice manipolare le informazioni provenienti dal modello al fine di generare l'output. La struttura del modello ad oggetti è particolarmente importante; se progettata bene, essa diventa una rappresentazione indipendente dall'output che potrà essere convertita facilmente in codice sorgente.

Exporter - accede all'Internal Object Model per prelevare informazioni necessarie al fine di generare codice. Tale modulo potrà usare template che guideranno il processo di generazione.

In Fig. 1 è mostrata l'architettura del generatore di codice. Essa è stata progettata pensando ai seguenti benefici:

- L'input è completamente indipendente dall'output in termini di tecnologie. Per esempio, si può creare un importer che legge UML ed exporter che scrivono codice C#/ADO e Java/JDBC a partire dallo stesso modello in input.
- Se l'IOM è implementato seguendo le regole dettate da UML, allora sarà possibile rappresentare tutto ciò che può essere fatto con UML. Ciò significa che qualsiasi tipo di input sarà convertito e rappresentato da un design orientato agli oggetti e questo è un gran vantaggio nella scrittura

degli importer e degli esporter.

- Implementando esporter multipli ed applicando un semplice pattern, sarà possibile generare differenti layer di codice (sorgenti, descrittori, script, documentazione, ecc...) con una singola passata. Questo assicura consistenza e sincronizzazione tra i diversi tipi di output.

PERCHÉ JAVA

Un tale ambizioso generatore di codice può essere efficacemente implementato usando uno dei moderni linguaggi di programmazione. Sarebbe troppo complesso implementarlo ricorrendo esclusivamente ad altre tecnologie come trasformazioni o linguaggi basati su template. Comunque, è anche vero che usare XSLT oppure un motore per template - tipo *Apache Velocity* - per specifici compiti potrà risultare più veloce ed efficace, specialmente quando si vuole generare codice relativo a semplici aspetti dei nostri progetti. Quello che stiamo proponendo è invece un generatore di codice sul quale deve essere possibile basare tutto il nostro progetto o almeno un parte consistente di esso. Sarà estremamente flessibile, intuitivo e basato su un potente modello ad oggetti. Quest'ultima caratteristica è spesso la maggiore limitazione degli approcci basati su trasformazioni o template; il modello ad oggetti è povero, rigido ed in alcuni casi completamente inesistente. Inoltre, usando un linguaggio imperativo sarà possibile trarre vantaggio da potenti feature, tool e librerie. Per esempio, scegliendo Java, abbiamo a disposizione caratteristiche come interfacce, reflection e gestione delle eccezioni che renderanno il generatore estensibile, affidabile e robusto.

IL MODELLO AD OGGETTI

Il modello interno del generatore è la parte più importante e delicata: una volta progettato non dovrebbe più subire cambiamenti sostanziali. Ciò che può essere importato dipende dalla corretta progettazione di questo modulo. Una buona strategia per implementare lo *IOM* è quella di *rubare* concetti presenti in UML; il modello può essere l'implementazione della struttura di un modello UML. Quindi, introdurremo un insieme di classi ed associazioni che rappresentano entità UML quali *classi, attributi, operazioni, parametri* e *associazioni*. Può suonare un po' strano, ma quello che stiamo facendo è progettare UML usando le regole di UML. Per convenzione, i nomi delle classi appartenenti al modello interno avranno il prefisso *IOM*. In Fig. 2 è presente il class diagram di un *IOM* notevolmente semplificato. La classe *IOMClass* modella una classe UML. Siccome le classi hanno attributi ed operazioni, *IOMClass* presenta aggregazioni con le classi *IOMAttribute* e *IOMOperation*. Quest'ultima classe ha anche un'aggregazione alla classe *IOMParameter,* che rappresenta i parametri di un'operazione. La classe *IOMAssociation* rappresenta un'associazione UML. Essa presenta due ruoli: *start* ed *end,* ovvero aggregazioni alla classe *IOMRole*. Quest'ultima è associata alla classe *IOMClass (involved)*. Tale corrispondenza indica la classe coinvolta nell'associazione. Sebbene il design dello *IOM* è notevolmente semplificato, esso dà comunque un'idea su come implementare uno *IOM* completo di tutte le funzionalità. In contesti reali, ogni classe avrà specifici attributi che in qualche modo guideranno la generazione del codice. Per esempio, per la classe *IOMClass* si potrebbero specificare i seguenti attributi: *nome, stereotype, visibilità, tipo, documentazione,* ecc... Inoltre, aspetti importanti come classi base e derivate, eccezioni e package dovranno essere supportati aggiungendo nuove classi ed associazioni al modello. Ecco l'implementazione della classe *IOMClass*:

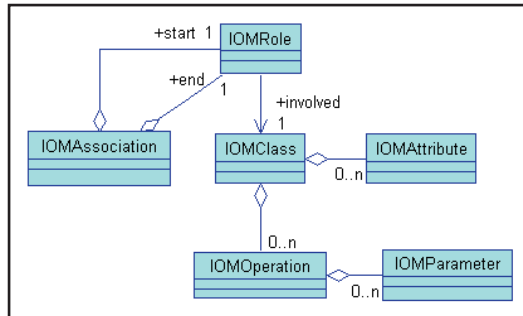
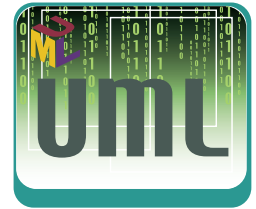
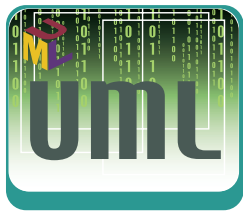



Fig. 2: L'Internal Object Model.

```
import java.util.*;
public class IOMClass {
  private String name = "";
  private String stereotype = "";
  private ArrayList attributes = new ArrayList();
  private ArrayList operations = new ArrayList();
  public String getName() {
    return name; }
  public void setName(String name) {
    this.name = name; }
  public String getStereotype() {
    return stereotype; }
  public void setStereotype(String stereotype) {
    this.stereotype = stereotype; }
  public ArrayList getAttributes() {
    return attributes; }
  public void addAttribute(IOMAttribute attribute) {
    attributes.add(attribute);
    attribute.setClassParent(this); }
  public ArrayList getOperations() {
    return operations; }
  public void addOperation(IOMOperation operation) {
    operations.add(operation);
```

Code Generation Network
http://www.codegeneration.net/

OMG Model Driven Architecture
http://www.omg.org/mda/

XDoclet
http://xdoclet.sourceforge.net/

Specifica XMI
http://www.omg.org/technology/documents/formal/xmi.htm

```
    operation.setClassParent(this); }
  public ArrayList getMyAssociations() {
    ArrayList ret = new ArrayList();
    for (int i = 0; i < IOMController.getAssociations()
                                              .size() ; i++)
      IOMAssociation ass = (IOMAssociation)
      IOMController.getAssociations().get(i);
      if(ass.getStartRole().getClassInvolved().
        getName().equals(this.name)
        ||ass.getEndRole().getClassInvolved().
        getName().equals(this.name))
        ret.add(ass); }
    return ret; }
}
```

Come si può notare, la classe presenta due attributi: *name* (il nome della classe) e *stereotype* (lo stereotype UML associato alla classe). Le aggregazioni con le classi *IOMAttribute* e *IOMOperation* sono implementate attraverso due *ArrayList* (*attributes* e *operations*) e relativi metodi *get/add*. La classe contiene inoltre il metodo *getMyAssociation* che restituisce tutte le istanze di *IOMAssociation* nelle quali la classe è coinvolta come nel ruolo start oppure in quello end. Adottando un simile approccio sarà possibile implementare le altre classi *IOM*.
La classe root del modello è *IOMController*, implementata come segue:

```
import java.util.*;
public class IOMController {
  private static ArrayList classes = new ArrayList();
  private static ArrayList associations = new ArrayList();
  public static ArrayList getClasses() {
    return classes; }
  public static ArrayList getAssociations() {
    return associations; }
  public static void addClass(IOMClass cl) {
    classes.add(cl); }
  public static void addAssociation(IOMAssociation
                                         association) {
    associations.add(association); }
  public static IOMClass queryClass(String name) {
    for (int i = 0; i < classes.size() ; i++) {
      IOMClass cl = (IOMClass) classes.get(i);
      if(cl.getName().equals(name))
        return cl; }
    return null;
  }
}
```

Questa classe consente di accedere alle istanze di *IOMClass* e di *IOMAssociation*. Inoltre, attraverso il metodo *queryClass*, sarà possibile ricercare un'istanza di *IOMClass* partendo dal nome della classe. *Importer* ed Esporter useranno *IOMController* rispettivamente per creare il modello interno e per leggere i metadati in modo da generare l'output.

IMPORTER: LEGGIAMO LO SCHEMA

L'importer ha la resposabilità di leggere il modello in input e creare lo IOM. Il generatore di codice avrà differenti importer per differenti sorgenti di dati. Secondo il paradigma MDA, l'importer legge un modello (generalmente esportato in XMI) che rappresenta un design UML creato con tool tipo Rational Rose o Genleware Poseidon. La nostra architettura è ancora più flessibile, infatti può supportare anche altri tipi di input come modelli proprietari e perfino codice sorgente.
Il concetto di importer può essere visto in Java come un'interfaccia. Essa può essere definita come segue:

```
public interface Importer {
  public void start() throws Exception;
  public IOMClass createClass(Object data) throws
                                              Exception;
  public IOMAttribute createAttribute(Object data)
                                        throws Exception;
  public IOMOperation createOperation(Object data)
                                        throws Exception;
  public IOMParameter createParameter(Object data)
                                        throws Exception;
  public IOMAssociation createAssociation(Object data)
                                        throws Exception;
  public IOMRole createRole(Object data) throws
                                              Exception;
}
```

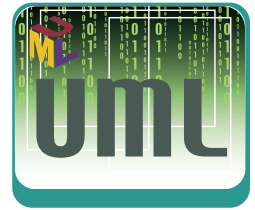
L'intefaccia contiene il metodo start che da l'inizio al processo di importing. Implementando i vari metodi *createXXX* sarà possibile creare la struttura IOM partendo dai dati in input. L'implemetazione dell'interfaccia Import dipende dal tipo di modello in input; quindi differenti modelli avranno differente implementazione. Nonostante questo, l'obiettivo comune ad ogni implementazione sarà quella di creare il modello ad oggetti. Di seguito è riportata un'implementazione parziale dell'interfaccia:

```
import java.util.*;
import javax.xml.parsers.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import java.io.*;
import java.util.*;
public class XMLImporter extends DefaultHandler
                                             implements
Importer{
  private InputStream input = null;
  private String fileName = null;
  private Stack stack = new Stack();
  public XMLImporter(String inputFileName) {
    this.fileName = inputFileName; }
  public void start() throws Exception {
```

BIBLIOGRAFIA

• **CODE GENERATION IN ACTION**
Jack Herrington
(Manning)
2003
http://www.manning.com/herrington/

• **CODE-GENERATION TECHNIQUES FOR JAVA**
Jack Herrington
(OnJava)
Settembre 2003
http://www.onjava.com/pub/a/onjava/2003/09/03/generation.html?page=1

• **GENERAZIONE AUTOMATICA DI CODICE**
G. Naccarato
ioProgrammo N. 79
2004.

```
  input = new FileInputStream(fileName);
  SAXParserFactory spf = SAXParserFactory.newInstance();
  spf.setNamespaceAware(true);
  SAXParser saxParser = spf.newSAXParser();
  XMLReader xmlReader = saxParser.getXMLReader();
  xmlReader.setContentHandler(this);
  xmlReader.parse(new InputSource(input));
  input.close(); }
 public IOMClass createClass(Object data) throws
                                         Exception {
  Attributes attr = (Attributes) data;
  IOMClass cl = new IOMClass();
  cl.setName(attr.getValue("name"));
  cl.setStereotype(attr.getValue("stereotype"));
  IOMController.addClass(cl);
  stack.push(cl);
  return cl; }
/* Implementare qui createAttribute, createOperation, ... */
 public void startElement(String uri, String name,
                         String qName, Attributes attr)
  throws SAXException {
  try {
   if (name.equals("Class")) {
    this.createClass(attr);}
   if (name.equals("Attribute")) {
    this.createAttribute(attr); }
   if (name.equals("Association")) {
    this.createAssociation(attr);}
   if (name.equals("Role")) {
    this.createRole(attr);}
   if (name.equals("Operation")) {
    this.createOperation(attr);}
   if (name.equals("Parameter")) {
    this.createParameter(attr); }
  } catch (Exception e) {
   e.printStackTrace();
   throw new SAXException(e); } }
 public void endElement(String uri, String name,
                    String qName) throws SAXException {
  if (stack.size()>0)
   stack.pop(); }
}
```

La classe *XMLImporter* opera su una semplice rappresentazione in XML di un class diagram UML. Tale formato può essere visto come un sott'insieme di XMI. Un esempio di modello può essere il seguente:

```
<?xml    version="1.0"       encoding="UTF-8"?>
<Content>
  <Class name="Customer" stereotype="">
    <Attribute name="code" type="integer"/>
    <Attribute name="description" type="string"/>
  </Class>
  <Class name="Order"       stereotype="">
    <Attribute name="number" type="integer"/>
    <Attribute name="date" type="date"/>
    <Operation name="getTotalAmount"
                                    returnType="double">
      <Parameter name="calculateVAT" type="boolean"/>
    </Operation>
  </Class>
  <Association name="">
    <Role class="Order" multiplicity="*" name=""
                                      type="start"/>
    <Role   class="Customer" multiplicity="1"
                              name="" type="end"/>
  </Association>
</Content>
```

Esso è una rappresentazione in XML del class diagram di Fig. 3. Come si può notare, la struttura del documento è molto intuitiva e richiama concetti UML quali classi, attributi, operazioni, parametri, associazioni e ruoli. La classe *XMLImporter* usa un parser SAX. Per ogni elemento trovato all'interno del file XML, essa invoca un opportuno metodo dell'interfaccia *Importer*. Per esempio, l'elemento *<class>* causerà l'invocazione del metodo *createClass*.
Questo metodo crea un nuovo *IOMClass* oggetto ed assegna un valore agli attributi name e stereotype secondo i dati provenienti dal documento XML. Gli altri metodi della famiglia *createXXX* possono essere facilmente implementati in modo simile.

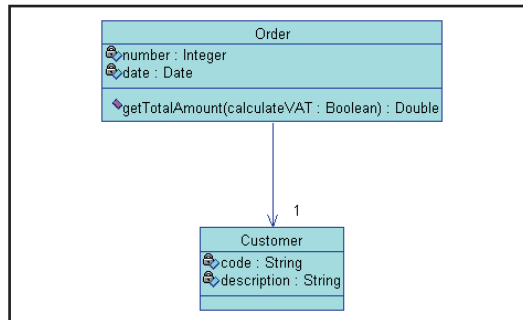
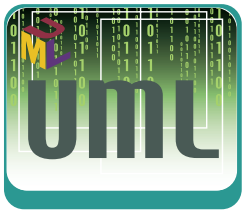



Fig. 3: ***Class Diagram per le classi Order e Customer.***

EXPORTER: ECCO IL CODICE

L'exporter converte il modello interno in un output specifico. Di seguito è riportata l'interfaccia Exporter:

```
import java.io.*;
import java.util.*;
public interface Exporter {
 public void start() throws Exception;
 public void initialize() throws Exception;
 public void startClass(IOMClass cl) throws Exception;
 public void endClass(IOMClass cl) throws Exception;
 public void startAssociation(IOMAssociation association,
                 IOMClass currentClass) throws Exception;
 public void endAssociation(IOMAssociation association,
                 IOMClass currentClass) throws Exception;
```

Giuseppe Naccarato è laureato in Scienze dell'Informazione e lavora a Glasgow (UK) come designer/ developer presso una multinazionale produttrice di software per dispositivi palmari. I suoi interessi sono orientati alle architetture distribuite basate su piattaforme J2EE e .NET. Può essere contattato attraverso la sua home page: www.giuseppe-naccarato.com

```
public void startOperation(IOMOperation operation)
                                    throws Exception;
public void endOperation(IOMOperation operation)
                                    throws Exception;
public void startAttribute(IOMAttribute attribute)
                                    throws Exception;
public void endAttribute(IOMAttribute attribute)
                                    throws Exception;
public void startParameter(IOMParameter parameter)
                                    throws Exception;
public void endParameter(IOMParameter parameter)
                                    throws Exception;
public void finalize() throws Exception;
}
```

L'architettura è molto simile a quella di un parser *SAX*. Il metodo start inizia la navigazione del modello ad oggetti, richiama il metodo *startClass* ogni qualvolta un oggetto *IOMClass* è processato ed in generale il metodo *startXXX* quando è processato un oggetto *IOMXXX*. I metodi *endXXX* sono invece invocati quando un'istanza di *IOMXXX* finisce di essere processata. Implementando l'interfaccia Exporter sarà possibile generare qualsiasi tipologia di output si desideri. Per esempio, di seguito è mostrata l'implementazione del metodo *startClass* che genera codice Java:

```
public void startClass(IOMClass cl) throws Exception {
  String fileName = cl.getName() + ".java";
  out = new PrintStream(new FileOutputStream(fileName));
  out.println("import java.util.*;");
  out.println("");
  String declaration = "public class ";
  out.println(declaration + cl.getName() + " {");
  out.println();
}
```

Il metodo crea un output file usando il nome della classe ed aggiunge l'estensione *.java*. Quindi, prelevando le informazioni dall'oggetto *IOMClass* in input, scrive il codice Java per dichiarare la classe. Implementando tutti i metodi dell'interfaccia sarà possibile ottenere la classe *JavaExporter* che crea classi Java che rispettano, in termini di attributi, operazioni e associazioni, il modello originale in input. Il programma principale del nostro generatore è il seguente:

```
import java.util.*;
public class Generator {
  public static void main(String args[]) throws Exception {
    if (args.length!=3) {
      System.out.println
        ("Sintax: Generator <input-file> <importer>
                                          <exporter>");
      System.exit(1); }
    Importer importer = null;
    if (args[1].equals("XML"))
      importer = new XMLImporter(args[0]);
    else {
      System.out.println("Importer " + args[1] + " non
                                            found!");
      System.exit(1); }
    Exporter exporter = null;
    if (args[2].equals("Java"))
      exporter = new JavaExporter();
    else {
      System.out.println("Exporter " + args[1] + " non
                                            found!");
      System.exit(1); }
    importer.start();
    exporter.start();
  }
```

L'applicazione presenta tre parametri da passare sulla linea di comando: il nome del file rappresentante il modello in input, un alias per l'importer da usare ed uno per l'esporter.
Lanciando l'applicazione:

```
> java Generator order.xml XML Java
```

saranno creati due classi Java nei file *Order.java* er *Customer.java* che seguiranno il modello presente nel file *order.xml*. La classe *Order* sarà la seguente:

```
/* Questo file è stato generato */
import java.util.*;
public class Order {
  private int number;
  public int getNumber() {return this.number;}
  public void setNumber(int number)
                              {this.number=number;}
  private Date date;
  public Date getDate() {return this.date;}
  public void setDate(Date date) {this.date=date;}
  private Customer customer;
  public Customer getCustomer() {return this.customer;}
  public void  setCustomer(Customer customer)
                          {this.customer=customer;}
  public double getTotalAmount(boolean calculateVAT) {
    /* ...method implementation... */
    return 0.0;
  }
}
```

CONCLUSIONI

In questo articolo abbiamo mostrato l'architettura, il design e l'implementazione in Java di un generatore di codice.
Nel prossimo articolo vedremo come migliorarlo utilizzando exporter multipli e template.

Giuseppe Naccarato

Automi riconoscitori

Una tipologia di algoritmi per string matching fa uso di automi deterministici a stati finiti; il problema del riconoscimento in generale, si arricchisce così, di significative ed interessanti tecniche.

Dopo aver trattato negli scorsi due appuntamenti il problema dello string matching ed avendo esaminato un buon numero di algoritmi risolutori è arrivato il momento di seguire un nuovo ed interessante filone che affronta la stessa questione in modo innovativo. Si tratta di esplorare quella che in generale è conosciuta come "pattern recognition", ma che nel caso specifico è semplicemente detta tecnica di riconoscimento, con l'ausilio di automi deterministici a stati finiti. Mostrerò l'algoritmo prototipo che risolve lo string matching; gli altri, che si basano su esso, e che sono più efficienti soprattutto su specifici campi di applicazione, verranno accennati in conclusione. Allo stesso tempo verrà presentato il modo in cui tali tecniche possano essere usate per il riconoscimento in generale. Importante, e spero interessante, si rileverà l'analisi e il conseguente sviluppo del software per l'implementazioni degli automi riconoscitori, il C++ ci aiuterà allo scopo.

AUTOMI DETERMINISTICI A STATI FINITI - DFA

Un automa deterministico a stati finiti, che da ora in avanti per sintesi indicheremo con la sigla anglosassone *DFA (Deterministic Finite Automation)*, è caratterizzato da una quaterna di soggetti. Con una formalizzazione sintatticamente esemplare un generico automa *A* è completamente descritto da due insiemi di stati, da uno stato iniziale e da un insieme di transizioni; così $A=(Q, q_0, T, E)$.
Esaminiamo singolarmente i diversi soggetti:

- *Q* è un insieme finito di stati;
- q_0 è lo stato iniziale;
- *T* è un sottoinsieme di *Q* e rappresenta l'insieme degli stati terminali;
- *E* è un sottoinsieme del prodotto cartesiano ($Q \times \Sigma \times Q$) e indica l'insieme delle transizioni.

L'accezione finito deriva dal fatto che tutti gli insiemi considerati sono finiti, ossia costituiti da un numero non infinito di elementi. *Q* è l'insieme degli stati che nella particolare applicazione del riconoscimento di stringhe corrisponde a tutti prefissi della stringa da ricercare. Lo stato q_0 è quello iniziale. *T* è un insieme di stati, raggiunti i quali, l'automa termina il suo compito. Nella ricerca coincide con un solo stato. Vedremo che un modo per trattare il DFA fa uso di grafi. In particolare, gli stati che presenta l'automa si associano a stringhe (prefissi della stringa da ricercare) e gli archi definiscono come un carattere, in un generico stato dell'automa, possa produrre una transizione verso un nuovo stato.
L'ultimo soggetto della quaterna, è un prodotto cartesiano tra l'insieme degli stati (considerati due volte) e l'alfabeto, indica i passaggi da uno stato ad un altro applicando uno degli elementi dell'alfabeto. Quest'ultimo indicato con Σ, rappresenta l'insieme dei simboli riconosciuti dal *DFA*. Per concludere la disquisizione teorica su cui si baseranno le nostre implementazioni è necessario definire il concetto di linguaggio applicato ad un automa. *L(A)* è un linguaggio *L* applicato al *DFA A* che produce una stringa *w* appartenente a Σ^* tale che, se esistono una serie di stati $q_0, q_1, \ldots, q_n$, con n pari alla lunghezza di *w* e con q_n appartenente a *T*, allora la transizione *(qi,w[i],qi+1)* appartiene a *E* con i compreso tra *0* e *n*. Si precisa che Σ^* indica tutte le possibili stringhe (parole) che si possono comporre a partire da i simboli costituenti l'alfabeto Σ. Nel prossimo paragrafo vedremo come tale modello matematico possa essere implementato e utilizzato per il problema oggetto delle nostra attuale analisi, ovvero il riconoscimento delle parole, anche conosciuto come string matching.

RICONOSCIMENTO UTILIZZANDO DFA

Una modalità chiara ed efficace per la descrizione di un *DFA* fa uso di grafi orientati. In tal caso ogni nodo è uno stato e ogni arco rappresenta un passaggio di stato (transizione che avviene applicando un simbolo dell'alfabeto). Vi sarà un solo stato iniziale e soltanto uno finale. Formalizziamo il *DFA* che verrà usato nello specifico problema di string-matching. La parola da trovare è indicata con x mentre il testo dove ricercarla è y. Si ha a che fare con un processo di parsing regolato dall'esplorazione di un grafo associato ad un automa. Il particolare *DFA* applicato alla stringa x, ovvero $A(x)$ è caratterizzato dalla seguente quaterna (Q, q_0, T, E) che abbiamo già descritto in modo generale nel precedente paragrafo. Nel caso specifico la caratterizzazione è la seguente:

- Q è l'insieme dei prefissi di x, ossia $Q=\{e, x[0], x[1]+x[2], \ldots, x[1]+x[2]+..+x[m-2],x\}$;
- $q_0=x$
- $T=\{x\}$
- Per q appartenente a Q (q è un prefisso di x) e a appartenente a S, (q, a, qa) appartiene ad E se e solo se qa è un prefisso di x, altrimenti (q, a, p) appartiene a E cosicché p è il più lungo suffisso di qa che è un prefisso di x.

Tale modello sarà il riferimento che utilizzeremo per lo sviluppo del programma. La definizione precedente non è altro che una generalizzazione della presente. Un esempio di *DFA* applicato allo string-matching chiarirà i concetti teorici esposti.

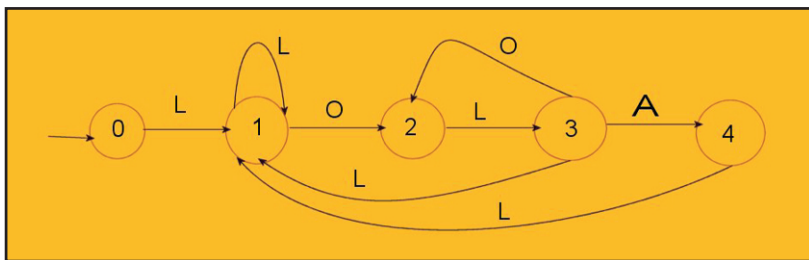



Fig. 1: ***Grafo corrispondente al DFA riconoscitore di stringhe.***

Supponiamo di voler cercare la stringa *"LOLA"* all'interno della sequenza *"PILLOLA POLO"*; lo so la frase non ha senso, non è necessario che ne abbia affinché funzioni. Vorrei comunque far notare che la parola cercata è presente come suffisso della prima parola pillola. Indicheremo con x il pattern (la parola da riconoscere) *"LOLA"* e con y il testo su cui effettuare tale ricerca *"PILLOLA POLO"*. Diamo forma adesso ai diversi soggetti del *DFA* specificando anche l'alfabeto a cui ci riferiamo.

```
Σ={A, I, O, L, P, bl}
Q={e, L, LO, LOL, LOLA}
q0=e
T={LOLA}
```

L'alfabeto per ovvi motivi di spazio è un sottoinsieme del comune alfabeto costituito da 26 simboli che conosciamo. Inoltre, tra i simboli riconosciuti dal *DFA* vi è lo spazio indicato con *bl*, che sta per blank. Q è l'insieme degli stati dell'automa che sono stringhe corrispondenti ai prefissi di x. Con e (minuscola) viene identificata la stringa vuota che corrisponde allo stato iniziale q_0. Il *DFA A* viene efficacemente rappresentato dal grafo riportato in Fig. 1.
Risulta che diverse stringhe x e y definite, su un alfabeto Σ, producono diversi automi riconoscitori. L'interpretazione del grafo è sottesa alla definizione del *DFA*, ad ogni modo esaminiamola. In qualsiasi stato ci troviamo a seconda del carattere dell'alfabeto che incontriamo passiamo in un diverso stato seguendo l'arco che con tale carattere indica la transizione. Qualora non sia riportato su grafo il carattere che abbiamo incontrato nel testo, la transizione genera automaticamente il passaggio allo stato iniziale. Quando si raggiunge uno stato finale (nella tipologia di *DFA* analizzati si ha un solo stato finale) il processo si conclude. Con riferimento all'esempio (grafo in Fig. 1) si parte dallo stato iniziale q_0. I primi due caratteri P e I non producono un passaggio di stato, come detto non essendo riportato alcun arco dallo stato q_0 con i simboli P e I si rimante in q_0. Il terzo carattere riporta un avanzamento dell'automa allo stato q_1. Successivamente, l'ulteriore presenza della lettera L, come indicato dall'arco che si avvita su se stesso, fa si che si permanga in q_1. Con la lettera O si passa allo stato q_2, poi con L allo stato q_3 ed infine il presentarsi della A conduce allo stato terminale q_4 che indica l'avvenuto riconoscimento del pattern. In questo ultimo passaggio un eventuale presenza della lettera O avrebbe portato allo stato q_2 in considerazione del fatto che si sarebbe composta già la sottostringa *LO*, a tale proposito si riesamini la definizione in quattro punti del *DFA* riconoscitore. Una volta raggiunto lo stato terminale (appartenente a T) il processo non si conclude, così come non si concluderà il programma che assoceremo a tale modello e, in conformità con le convenzioni adottate per lo string matching, verranno ricercati ulteriori parole. Nell'esempio la presenza del blank porta allo stato iniziale, successivamente il processo si evolve alla ricerca di nuovi match che in un caso specifico non avverranno.

SVILUPPO DEL PROGRAMMA

Il passo successivo è sviluppare un programma risolutore; per farlo saranno necessarie tre fasi: la definizione di un'opportuna struttura dati, la produzione

NOTA

RIFERIMENTI SU IOPROGRAMMO

Riferimenti ad altri articoli della sezione soluzioni della rivista si trovano ai numeri 61 e 62 per approfondimenti circa gli automi e ai due numeri scorsi, 78 e 79 per il problema dello string matching in generale.

di routine di base e infine lo sviluppo del programma costruito sulla base dei risultati, ottenuti nelle due fasi precedenti. Seguendo una logica top down partiremo con l'ultima delle fasi. La qualità di linguaggio strutturato del C++, ci consentirà di produrre successivamente le routine di base e di definire un'opportuna struttura dati. Per rendere il codice più comprensibile e al fine di dedicarci essenzialmente al metodo, concentreremo la nostra attenzione allo sviluppo delle routine; chi volesse può facilmente produrre la versione object oriented costruendo classi basate sul codice prodotto. La prima routine da fare si occupa di costruire l'automa aut a partire dalla stringa *x* di lunghezza *m*. Ecco il codice:

```
void preAut(char *x, int m, Graph aut) {
  int i, state, target, oldTarget;
  for (state = getInitial(aut), i = 0; i < m; ++i) {
    oldTarget = getTarget(aut, state, x[i]);
    target = newVertex(aut);
    setTarget(aut, state, x[i], target);
    copyVertex(aut, target, oldTarget);
    state = target;
  } setTerminal(aut, state);
}
```

Una volta costruito il *DFA* bisogna sottoporre la stringa *x* di lunghezza *m* al testo *y* di lunghezza *n*. Tale elaborazione avverrà in due distinte fasi. In una prima elaborazione si predispone l'automa alla ricerca, successivamente mediante un ciclo che scandisce tutti gli *n* elementi del testo *y* e partendo dallo stato iniziale si effettua la ricerca vera e propria. Ad ogni iterazione del ciclo si ottiene, mediante la routine *getTarget*, il nuovo stato che assume l'automa *aut*, che si trova nello stato state e a cui viene sottoposto il generico carattere *y[j]*. Se lo stato ottenuto appartiene all'insieme dei terminali si conclude la procedura con il relativo output, altrimenti si continua con nuove iterazioni del ciclo.

```
void AUT(char *x, int m, char *y, int n) {
  int j, state;
  Graph aut;
  /* PreElaborazione */
  aut = newAutomaton(m + 1, (m + 1)*ASIZE);
  preAut(x, m, aut);
  /* Ricerca */
  for (state = getInitial(aut), j = 0; j < n; ++j) {
    state = getTarget(aut, state, y[j]);
    if (isTerminal(aut, state))
      OUTPUT(j - m + 1);}
}
```

Dall'analisi della complessità dell'algoritmo si rileva che la prima routine di memorizzazione dell'automa consta di $O(m + s)$ se la memorizzazione avviene in una tabella ad accesso diretto. Per la seconda procedura la fase di pre-elaborazione ha anch'essa una complessità $O(m \times s)$. Infine, la ricerca vera e propria ha un costo computazionale $O(n)$ se l'automa è memorizzato in tabella ad accesso diretto, altrimenti $O(n \times \log(s))$. Il valore *s* è costante.

STRUTTURA DATI E ROUTINE DI BASE

Come accennato in precedenza l'intera struttura dati si basa su una lista a puntatori. Ecco la dichiarazione di tale record *(struct)* dinamico.

```
struct _cell{
  int element;
  struct _cell *next; };
typedef struct _cell *List;
```

NOTA

GRAFI

I grafi sono strutture reticolari costituite da nodi e archi questi ultimi collegano tra loro due nodi e possono essere orientati, ovvero sono dotati anche di verso. Possono essere rappresentati o in modo grafico come ad esempio in Fig 1, o attraverso apposite strutture dati. Una prima soluzione prevede una struttura dati con elementi dinamici, ossia, di liste a puntatori che si avvicinino alla rappresentazione grafica. Una seconda con matrici di transizione di stato e transizione di uscita che mettono in relazione gli ingressi in funzione rispettivamente gli stati stessi e le uscite. Nel primo caso gli elementi della matrice sono le transizione di stato. Infine, un'altra possibilità è quella di schematizzare il grafo sempre con matrici, in tal caso si producono le matrici di incidenza e di adiacenza.

Il grafo sarà una struttura più complessa in cui si mantengono tutte le informazioni necessarie alla costruzione e manipolazione dello stesso, come il numero di archi *(edgeNumber)*, il numero di vertici, e particolari nodi quali gli stati iniziali, i terminali, i target su suffissi e la posizione corrente.

```
struct _graph {
  int vertexNumber,
      edgeNumber,
      vertexCounter,
      initial,
      *terminal,
      *target,
      *suffixLink,
      *length,
      *position,
      *shift;  };
 typedef struct _graph *Graph;
 typedef int boolean;
```

Le routine di manipolazione sono diverse. Sono di seguito presentati tutti i prototipi, il che rende il programmatore lettore libero di implementarli nella maniera che desidera. In coda al paragrafo, ad ogni

modo, viene proposta la soluzione che fa uso di strutture dinamiche che per ovvi motivi di spazio è riferita a una routine a campione.

```
/* Per un grafo restituisce una nuova struttura dati
                                con v vertici ed e archi */
Graph newGraph(int v, int e);
/* Restituisce una struttura dati per un automa
                                 con v vertici ed e archi*/
Graph newAutomaton(int v, int e);
/* Restituisce per automatizzare i suffissi con v vertici
                                           ed e archi  */
Graph newSuffixAutomaton(int v, int e);
/* Restituisce una nuova struttura dati per alberi
                                con v vertici ed e archi */
int newVertex(Graph g);
/* Restituisce i vertici iniziali del grafo g */
int getInitial(Graph g);
/* Restituisce il valore logico vero se v è un vertice
                                  terminale del grafo g */
boolean isTerminal(Graph g, int v);
/* Imposta i vertici terminali del grafo g */
void setTerminal(Graph g, int v);
/* Restituisce i target (destinazione) degli archi dal vertice
              v in corrispondenza del carattere c nel grafo g */
int getTarget(Graph g, int v, unsigned char c);
/* Aggiunge l'arco dal vertice v al vertice t in
               corrispondenza del carattere c nel grafo g */
void setTarget(Graph g, int v, unsigned char c, int t);
/* Restituisce il link al suffisso del vertice v nel grafo g */
int getSuffixLink(Graph g, int v);
/* Imposta i link ai suffissi del vertice v al vertice s nel
                                                 grafo g */
void setSuffixLink(Graph g, int v, int s);
/* Restituisce la lunghezza del vertice v nel grafo g */
int getLength(Graph g, int v);
/* Imposta la lunghezza del vertice v all'intero ell
                                              nel grafo g*/
void setLength(Graph g, int v, int ell);
/* Restituisce la posizione del vertice v nel grafo g */
int getPosition(Graph g, int v);
/* Imposta la lunghezza del vertice v all'intero ell
                                             nel grafo g */
void setPosition(Graph g, int v, int p);
/* Restituisce lo spostamento dell'arco dal vertice v
                     per mezzo del carattere c nel grafo g */
int getShift(Graph g, int v, unsigned char c);
/* Imposta lo spostamento dell'arco dal vertice v per
                        mezzo del carattere c nel grafo g */
void setShift(Graph g, int v, unsigned char c, int s);
/* Copia tutti i caratteri dal vertice sorgente al vertice
                                         target nel grafo g */
void copyVertex(Graph g, int target, int source);
```

Al fine di comprendere le singole routine sono state opportunamente commentate secondo la sintassi C++. A titolo di esempio ecco la possibile implementazione di un paio delle procedure presentate come prototipo.

```
Graph newGraph(int v, int e) {
  Graph g;
  g = (Graph)calloc(1, sizeof(struct _graph));
  if (g == NULL)
    error("newGraph");
  g->vertexNumber  = v;
  g->edgeNumber    = e;
  g->initial       = 0;
  g->vertexCounter = 1;
  return(g); }
Graph newAutomaton(int v, int e) {
  Graph aut;
  aut = newGraph(v, e);
  aut->target = (int *)calloc(e, sizeof(int));
  if (aut->target == NULL)
    error("newAutomaton");
  aut->terminal = (int *)calloc(v, sizeof(int));
  if (aut->terminal == NULL)
    error("newAutomaton");
  return(aut); }
```

La prima costruisce un nuovo grafo, impostando il valore iniziale di alcune variabili (numero di vertici, numero di archi, etc.). La seconda invece, a partire dal grafo costruito, associa un *DFA* individuando vertici e nodi, iniziali e terminali e riservando opportunamente la memoria. Nel caso specifico è stata usata la sintassi tipica del C, riconosciuta ovviamente da C++.

CONCLUSIONI

Esistono molti altri metodi che basano la ricerca sulla costruzione e l'uso di un *DFA*. Tra questi si ricordano l'algoritmo *Simon* usato soprattutto nell'ambito dell'economia che costruisce un grafo in cui si riducono al minimo il numero di archi; in particolare il numero limite di tali archi è *2m*. Per questo algoritmo la transizione su alcuni stati, anche se non memorizzata, può essere dedotta a partire dal percorso pregresso. Altri algoritmi conosciuti che usano grafi sono:

- Fattore inverso
- Turbo fattore inverso
- Dawg Matching in avanti
- Dawg Matching non deterministico all'indietro
- Oracle Matching all'indietro

Spero che l'argomento che si può considerare concluso, a meno di approfondimenti, sicuramente non prossimi, possa avervi interessato avendo aggiunto importanti nozioni circa il delicato quanto importante tema del riconoscimento. Alla prossima.

Fabio Grimaldi

Approfondiamo un tipico problema matematico

Il problema Knapsack e programmazione dinamica

di Fabio Grimaldi

Esistono molti modi per risolvere il problema knapsack: uno dei più interessanti risulta essere possibile con la programmazione dinamica. Di seguito trovate il completo algoritmo e la relativa implementazione.

Nel problema *knapsack*, viene ricercata la migliore soluzione per riempire un sacco di fissata capienza con un numero finito di oggetti, ognuno dei quali ha un valore e un ingombro. In definitiva si vuole scegliere un insieme di oggetti da riporre nel sacco al fine di massimizzare il valore e rispettare la capienza massima del sacco. Tale problema è conosciuto in letteratura informatica anche come il problema del ladro che si trova a far fruttare al massimo il suo furto rispettando i limiti di spazio del suo zaino.

FORMALIZZAZIONE

Formalizzando scriviamo come nello scorso appuntamento. Sia S un insieme di oggetti $S=\{1,2,..n\}$ con d_i e v_i rispettivamente la dimensione (o peso a seconda della divulgazione) e il valore del oggetto *i-esimo;* ovviamente, i è compreso tra 1 e n. Si vuole individuare un sottoinsieme di S, che indicheremo con S' costituito da m oggetti di S con $m<=n$ in modo che si massimizzi:

$$max \sum_{j \in S'} v_j$$

con $|S'|=m$. Devono essere rispettati i vincoli di spazio; essendo C la capacità del sacco, deve accadere che:

$$\sum_{j \in S'} d_j \leq C$$

L'accezione 0/1 che si trova in letteratura associata a Knapsack problem indica la variante più diffusa, e cioè quella che prevede una logica binaria rispetto ad ogni oggetto, ossia se è presente o meno nel sacco (uno o zero). Qualora sia lecito considerare una parte di oggetto si ha a che fare con la variante frazionaria di Knapsack problem. Una prima soluzione prevede l'esplorazione di tutte le soluzioni. Si tratta di individuare tutte le combinazioni di ampiezza m che si possono riscontrare (ancora una volta un puro problema di calcolo combinatorio) e scegliere quella che massimizzi il valore. Tale soluzione presenta una complessità elevata considerata la natura esponenziale del calcolo combinatorio da effettuare, più volte ribadito tra queste pagine. Più precisamente si riscontra $O(2^n)$.

IMPLEMENTAZIONE

Mediante programmazione dinamica proviamo a ricercare una soluzione efficiente. Il nodo della questione è individuare ed esplorare in modo opportuno i sottoinsiemi di dimensione m, possibili soluzione ottime. In conformità ai nomi delle variabili usate nella formulazione del problema come struttura dati usiamo un vettore d per i pesi e uno v per valori. La variabili n indica il numero totali di elementi. La dimensione m del sottoinsieme ottimo non conta in fase di sviluppo dell'algoritmo. Inoltre, i due indici i e k rispettivamente generalizzeranno C e n, e verranno usate nelle istruzioni cicliche per scorrere negli intervalli che tali valori definiranno. Il metodo che usiamo è definito ricorsivamente.
Si fa uso di una matrice che raccoglie i dati parziali e che al termine dell'elaborazione conterrà anche la soluzione; di nome *sp.* Ecco come si ottiene ricorsivamente un generico valore di *sp.* Esso è assegnato in funzione della condizione $d[k]>i$ come segue.

```
sp[k, i]=sp[k-1, i]    se d[k]>i
sp[k, i]=max(sp[k-1,i], sp[k-1,i-d[k]]+ v[k] )
                                    negli altri casi
```

La riga k della matrice indica che stiamo elaborando la soluzione con k oggetti nel sacco, cosicché siamo in grado di trovare le soluzioni ottime per ogni k. L'implementazione della definizione matematica in codice pascal è a questo punto scontata.
Tralasciando l'ovvia dichiarazione e l'input dei vettori dei pesi e dei valori, nonché le altre

grandezze come *C*, e *n*; possiamo scrivere:

```
…
for i:=0 to C do
    sp[0,i]:=0;
for k:=0 to n do
Begin
    sp[k,0]:=0;
    for i:=0 to C do
      if d[k]<=i then (* Può far parte della soluzione
*)
           if (v[k]+sp[k-1, i-d[k]]>sp[k-1,i]) then
             sp[k,i]:=v[k]+sp[k-1,i-d[k]]
      else sp[k,i]:=sp[k-1,i]
           else sp[k,i]:=sp[k-1,i] (*Se d[k]>i *)
End
…
```

Il ciclo su *k* individua le soluzioni ottime per ogni ampiezza del sottoinsieme di soluzioni (numero di oggetti nel sacco). Il ciclo interno su *i* scorre fino ad arrivare alla capienza del sacco, ad ogni iterazione verifica se un generico oggetto può far parte della soluzione; in caso affermativo mediante un nuovo controllo, trova la soluzione più vantaggiosa in termini di valori. L'analisi della complessità evidenzia che la prima fase di azzeramento consta di $O(C)$ così come il secondo *for* su *i*. Questo ultimo ciclo si trova innestato in un altro ciclo ci n iterazioni. Quindi l'intero blocco sarà $O(n*C)$ che ovviamente risulta essere un risultato migliore di $O(2^n)$. Ad ogni modo non è apprezzabile in termini computazionali il fatto che la complessità dipenda da *C* (ampiezza del sacco), questo porta ad una significativa dipendenza delle prestazioni dell'algoritmo dai dati usati.
Altri metodi invece non soffrono tale fattore negativo.

L'ANGOLO DELLA COMPETIZIONE

IL PROBLEMA DELLE LAMPADE

Ricordo cosa da questo angolo era stato proposto.

Per illuminare un salone si hanno a disposizione un set di N lampade colorate e numerate da 1 a N. Le lampade sono connesse a 4 pulsanti:

- **PULSANTE 1: quando viene premuto tutte le lampade cambiano il loro stato; quelle spente si accendono e quelle che sono accese si spengono.**
- **PULSANTE 2: cambia lo stato delle lampade che hanno numero dispari.**
- **PULSANTE 3: cambia lo stato delle lampade che hanno numero pari.**
- **PULSANTE 4: cambia lo stato delle lampade che hanno numero nella forma 3k+1 con 0<=K, quindi 1, 4, 7,… .**

C'è un contatore C che registra il numero totali di pressioni di bottoni. Quando la festa comincia tutte le lampade sono accese e il contatore C ha valore zero. Vi viene fornito il valore del contatore C e varie informazioni sullo stato finale di alcune lampade. Si richiede di scrivere un programma che determini tutte le possibili configurazioni finali delle N lampade in accordo con le informazioni fornite, senza ripetizioni. Gli input e gli output sono rispettivamente contenuti in input.txt e output.txt. Il primo dei due file contiene sulla prima riga il singolo intero N e sulla seconda il valore del contatore C. La terza riga contiene la lista delle lampade che sicuramente sono accese nella configurazione finale, mentre la riga sottostante (la quarta) contiene la lista dei numeri di lampade che sicuramente sono spente nella configurazione finale. Non si possiedono informazioni sulle lampade il cui numero non compare in nessuna delle due liste. Queste potranno quindi essere accese o spente, nella configurazione finale. Per la terza e la quarta riga vale la regola che i numeri delle lampade che compaiono nella lista sono separati da spazio e terminati da un -1. Il file output.txt, deve contenere tutte le possibili configurazioni delle lampade che rispettano le informazioni fornite nel file di input, senza ripetizioni. L'ordine con il quale le configurazioni vengono elencate non è rilevante.
Ogni riga del file di output deve contenere N caratteri, senza separatori. Il carattere i-esimo rappresente lo stato dell'i-esima lampada in quella particolare configurazione e può essere uguale solo a 1 (che indica che la lampada è accesa) o 0 (lampada spenta).
Sintetizzando, per motivi di spazio, con il procedimento proposto nel sito ufficiale delle olimpiadi di informatica, in una prima fase di analisi deduciamo come dalla conoscenza dello stato di alcune lampade si possa dedurre lo stato di altre. In particolare, concentrandoci sulle regole 2, 3 e 4 di passaggio di stato, conoscendo la lampada 2 si dedurranno tutte quelle pari, eccetto quelle che rientrano nella forma 3k+1 (4, 10, 16 …) analogamente, con la 3 tutte le dispari eccetto (7, 13 ..).
Infine conoscendo lo stato della L 1 e quelle dispari non nella forma 3k+1, si può dedurre lo stato delle lampade dispari nella forma 3k+1; analogo discorso per le pari. Il pulsante 1 influenzando tutte le lampade è come se di fatto non ne influenzasse nessuna. Sul sito web di ioProgrammo (nella sezione download codice del numero 80) si riporta la soluzione come codice pascal (*intelligiochi.zip*) e si rimanda alla prossima puntata per ulteriori spiegazioni del problema.
Chi ne ha prodotto una propria può fare dei confronti anche in termini di complessità

Biblioteca

ON LINE

ONJAVA

Articoli, Blog, link, news...una miniera di informazioni per gli sviluppatori Java.

http://www.onjava.com/

C# HELP

Una impressionante raccolta di articoli, in lingua inglese, per il nuovo linguaggio Microsoft dedicato alla piattaforma .NET.

http://www.csharphelp.com/

VB.NET HEAVEN

Così come si evince dalla sua url: il paradiso per gli sviluppatori Visual Basic .NET; articoli, codice sorgente, news, link a iosa.

http://www.vbdotnetheaven.com/

JGURU

Il cuore pulsante di questo sito è il forum: ricco di suggerimenti e risposte a tutti i dubbi che potrebbero mai sorgere nella mente di un programmatore Java. Da leggere con attenzione anche le numerose Faq e gli ottimi articoli proposti.

http://www.jguru.com/

ASP.NET 2.0 REVEALED

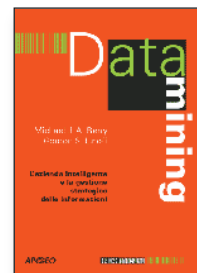
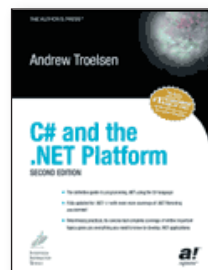
Un testo che permette di scoprire tutti i segreti del linguaggio che venne rivelato al pubblico, per la prima volta, come parte dell'architettura .NET Framework, al Microsoft PDC, svoltosi nell'ottobre 2003. L'autore svela, dal punto di vista dello sviluppatore, tutte le nuove ed interessanti caratteristiche del linguaggio ASP.NET. Naturalmente, per leggere e comprendere appieno gli argomenti trattati, è richiesta una conoscenza di base della versione 1.1 del linguaggio per poter, così, procedere, allo step d'apprendimento successivo. I principali argomenti trattati:

- Data Controls avanzati
- Master Pages
- Site Navigation
- Creazione di portali con Web Parts
- Page Frameworks avanzati.

Difficoltà: Media • Autore: Patrick A. Lorenz • Editore: APress http://www.apress.com
ISBN: 1-59059-337-5 • Anno di pubblicazione: 2004 • Lingua: Inglese
Pagine: 416 • Prezzo: $ 39.99

DATA MINING

Grazie a questo testo scoprirete come esplorare ed analizzare un'ampia mole di dati, al fine di scoprire regole e modelli significativi per poterli organizzare. Si sa, l'informazione è l'asset strategico di ogni azienda, ecco perché è importante conoscere i metodi e le tecnologie di data mining; esse consentono di estrarre dai giacimenti di dati tutto il loro potenziale in termini di efficacia. Dopo la trattazione dei fondamenti della raccolta dati e del Customer Relationship Management, sono forniti suggerimenti, strategici e tattici e sono illustrati casi pratici, in settori quali l'e-commerce, il bancario e le telecomunicazioni. Alcuni degli argomenti trattati: Metodologie e tecnologie per il data mining, Consolidamento e fidelizzazione della base clienti, Identificazione dei target e Ottimizzazione delle strategie, Individuazione di nuovi mercati per prodotti e servizi, opprtunità di cross-selling.

Difficoltà: Bassa • Autori: Michael J.A. Berry – Gordon S. Linoff
Editore: Apogeo http://www.apogeonline.com **• ISBN: 88-7303-865-4**
Anno di pubblicazione: 2001 • Lingua: Italiano • Pagine: 608 • Prezzo: € 39.00

C# AND THE .NET PLATFORM – 2° ED

L'upgrade di un libro che ha riscosso un grandissimo successo, tradotto in ben otto lingue. Indirizzato agli sviluppatore che hanno già avuto qualche esperienza con i linguaggi ad oggetti.
Dopo una breve panoramica sul linguaggio C#, questo testo illustra gli aspetti chiave della piattaforma .NET. L'autore mostra ai programmatori gli strumenti necessari per trarre vantaggio dal .NET remoting protocol. Un'ampio sguardo viene dato anche sull'utilizzo di C# per le applicazioni GUI. Il testo rappresenta un perfetto equilibrio tra la trattazione di C# e di .NET, ed inoltre una guida al Web development, ai Web services e all'accesso dati con ADO.NET.
Dalle applicazioni Windows-based a quelle Web-based, in unico ed imperdibile testo.

Difficoltà: Medio-Alta • Autore: Andrew Troelsen • Editore: Apress
http://www.apress.com **• ISBN: 1-59059-055-4 • Anno di pubblicazione: 2003**
Lingua: Inglese • Pagine: 1200 • Prezzo: $ 59.99

JavaScript Kit

JavaScript è oggi una realtà di fatto ben consolidata e assai nota ai Web-master coinvolti nel settore del Web-publishing.

JavaScript è un linguaggio di scripting progettato e sviluppato a quattro mani da Netscape e Sun Microsystems, che venne implementato per la prima volta su Netscape Navigator 2.0. Un linguaggio di scripting è sostanzialmente un linguaggio di programmazione in qualche modo ridotto, pensato non tanto per avere vita propria, quanto piuttosto per estendere le possibilità di un software o di un'altra tecnologia. In quest'ottica possiamo considerare JavaScript come un'estensione di HTML. Quando JavaScript si affacciò sul Web, la novità fu piuttosto impressionante: era finalmente possibile vedere delle pagine HTML la cui dinamicità andasse oltre le semplici GIF animate. Questo bastò per consentire a JavaScript una velocissima ed ampia diffusione, tanto che Microsoft stessa, iniziando a mostrare interesse per Internet ed avendo fallito il tentativo di VBScript, si ritrovò a dover piegare le ginocchia e ad includere in Internet Explorer 3.0 un linguaggio assai simile a JavaScript che, per motivi di copyright, fu battezzato JScript. Oggi, a distanza di circa otto anni da quegli avvenimenti, JavaScript e JScript viaggiano ufficialmente di pari passo, senza però mai smettere di presentare agli occhi degli sviluppatori alcune sostanziali differenze che, data l'importanza della portabilità di un codice nell'ambito di Internet, risultano essere più un freno che una maggiore possibilità di scelta. JavaScript, ad ogni modo, è stato standardizzato da un ente europeo chiamato ECMA, operazione che ha portato alla nascita dello standard ECMAScript. Altri standard del W3C, come DOM, permettono infine la possibilità di stesura di applicazioni DHTML completamente compatibili con i browser di più recente emissione.

JAVASCRIPT NON È JAVA

Un concetto molto importante da chiarire è la differenza che corre tra JavaScript e Java. Questi due linguaggi, infatti, condividono ben poco all'infuori del nome. JavaScript non assomiglia a Java più di quanto non assomigli a C o a C++. Insomma, è bene non confondere mai i due linguaggi, che si differenziano tanto per i contenuti quanto per gli scopi. Il nome JavaScript è semplicemente stato imposto da una questione di marketing, tanto che la primissima versione del linguaggio progettata dalla sola Netscape si chiamava *LiveScript*, in parallelo con un'altra tecnologia sempre di Netscape denominata *LiveWare* e destinata all'ambiente dei server.

Fig. 1: La homepage del Sito.

PERCHÉ IMPARARE JAVASCRIPT

Sia ben chiaro: si possono realizzare ottimi siti Web anche senza far uso di JavaScript. Anzi, in molti casi capita di vedere siti che usano JavaScript sino all'eccesso, per inutili vezzi privi di senso e di usabilità. Nonostante questo, il perché originario di JavaScript permane: l'interazione lato client con l'utente può essere utile e proficua. Pensiamo ai casi in cui JavaScript viene impiegato per la validazione preliminare dei moduli, per i rollover sulle immagini (un vezzo di per sé non indispensabile ma che può comunque essere sfruttato assai gradevolmente), per la gestione di finestre aggiuntive e così via. Inoltre JavaScript non è strettamente legato a HTML. Data la sua diffusione, lo standard ECMAScript viene oggi preso a riferimento ogni volta che un ambiente deve essere ampliato con caratteristiche di scripting. Si pensi ad SVG, lo standard di W3C per la grafica vettoriale di cui si è parlato in questa rubrica il mese scorso, che alla fine è un insieme di XML, CSS e JavaScript. Numerose applicazioni, inoltre, usano JavaScript come linguaggio di scripting interno. Le interfacce XUL di Mozilla, ad esempio, possono essere programmate via JavaScript. Abbandonando il lato client, c'è da constatare che JavaScript si è ricavato una buona posizione anche sul lato server: le pagine ASP, ad esempio, possono essere programmate sfruttando JScript. Microsoft, inoltre, ha da principio incluso in .NET una nuova versione di JScript (battezzata JScript.NET). Imparare JavaScript può allora essere interessante e proficuo, sia nel caso in cui il proprio sguardo sia rivolto allo sviluppo Web sia in altre situazioni.

JAVASCRIPT KIT

JavaScript Kit è un sito completamente dedicato al mondo di JavaScript, con un particolare occhio di riguardo ai suoi utilizzi nell'insieme di DHTML. Sono molte le risorse che potrete trovare all'interno del sito in oggetto. Si comincia da una lunga serie di JavaScript pronti all'uso, suddivisi per categorie. Troverete orologi, calendari, scroller, menu, effetti per il testo, giochi, modelli per la validazione e così via. Per chi fosse interessato all'apprendimento di JavaScript, il sito mette a disposizione una nutrita schiera di tutorial (in lingua inglese, come il sito stesso), anche questi suddivisi per categorie. Potrete così imparare a realizzare da voi le più gettonate applicazioni JavaScript, come i rollover, il riconoscimento del browser in uso, la manipolazione delle stringhe, la validazione dei moduli e via discorrendo. In due macro-sezioni differenti, JavaScript Kit propone tutorial per principianti e tutorial di stampo avanzato. In questa seconda categoria troverete articoli molto interessanti sulle espressioni regolari e sulla programmazione orientata agli oggetti. Insieme ai tutorial sul linguaggio, JavaScript Kit propone inoltre una raccolta di articoli incentrati sulle tecnologie ad esso collegate, come CSS, HTML e DHTML. Per non deludere chi fosse finito su JavaScript Kit per errore mentre era alla ricerca di materiale Java destinato al Web, viene proposta anche una raccolta delle più famose applet Java. Il sito è abbastanza ben fatto e si lascia navigare facilmente; il motore di ricerca interno funziona sufficientemente bene. Se siete interessati all'argomento, questa è una visita che conviene tentare.

Carlo Pelliccia

INBox
L'esperto risponde...

API e WORD

Ciao ragazzi, ho un piccolo problema con un 'applicazione , questa applicazione in base a un'utenza gestisce l'apertura di file Word disabilitando o abilitando determinate cose.
Per adesso ho usato il riferimento in vb a microsoft word object library e l'applicazione funziona , ma se c'è una versione precedente di word mi apre word ma non mi riesce a aprire il documento che gli passo e non mi abilita/disabilita quello che gli passo.
Il mio codice è questo:

```
Public Sub ControlloWord()
Dim WApp As Word.Application
Set WApp = New Word.Application
WApp.Visible = True
If frmMain.StatusBar1.Panels(3).Text
        = "LIVELLO 1" Then
WApp.Documents.Open
frmDocumento.
        txtPercorso.Text, , True
WApp.CommandBars.ActiveMenuBar.
        Enabled = True
WApp.CommandBars("Standard"
        ).Enabled = True
ElseIf frmMain.StatusBar1.Panels(
        3).Text = "LIVELLO 2" Then
WApp.Documents.Open
frmDocumento.
        txtPercorso.Text, , True
WApp.CommandBars.ActiveMenuBar.
        Enabled = True
WApp.CommandBars("Standard").
        Enabled = True
ElseIf frmMain.StatusBar1.Panels(
        3).Text = "LIVELLO 3" Then
WApp.Documents.Open
frmDocumento.
        txtPercorso.Text, , True
WApp.CommandBars.ActiveMenuBar.
        Enabled = False
WApp.CommandBars("Standard"
        ).Enabled = False
ElseIf frmMain.StatusBar1.Panels(
        3).Text = "LIVELLO 4" Then
WApp.Documents.Open frmDocumento
        .txtPercorso.Text, , True
WApp.CommandBars.ActiveMenuBar.
        Enabled = False
WApp.CommandBars("Standard").
        Enabled = False
End If
End Sub
```

ora la domanda: non c'è un altro metodo o un'API che funzioni indipendentemente dalla versione di WORD? Ciao

Aristotele

Risponde Ivan Venuti

Ciao, credo che il problema risieda proprio nel fatto che tu usi l'associazione anticipata (early binding): dovresti usare quella tardina (late binding). Mi spiego: tu hai dichiarato un riferimento alla libreria di oggetti Word e grazie a questo riferimento puoi usare gli oggetti Word come se fossero parte degli oggetti che usi di solito (e hai il vantaggio del completamento automatico del codice e "buone" prestazioni in termini di velocità). Ora quello che scrivi presuppone che sulla macchina su cui andrai ad installare il programma sia presente Word e (PRESUMO) anche la stessa versione della libreria. Dico PRESUMO in quanto non l'ho mai verificato avendo cura di sviluppare con versioni di Word uguali a quelle del cliente. Il modo per superare tali limitazioni (assenza di Word o versione diversa) è usare il late binding (ovviamente se Word non esiste dara' un errore al run time ma che puoi comunque intercettare e gestire). Nel concreto il late binding fa uso della funzione CreateObject("nomeOggetto"); nel tuo caso:

```
Dim WApp As Object
Set WApp = CreateObject("Word.Application")
...
```

Ovviamente usando il late binding perdi tutte le info sugli oggetti che avevi al compile time. Ti suggerisco pertanto di sviluppare le applicazioni usando l'early binding e poi passare al late binding quando il codice è stabile e pronto per l'installazione. Dimenticavo: devi anche fare attenzione che gli oggetti che usi (ed eventuali metodi/proprietà) siano presenti nella versione di Word che ha l'utente finale. Infatti certi costrutti sono presenti a partire da una determinata versione e non nelle precedenti.
Dal tuo codice, non mi pare che tu usi oggetti particolari quindi non dovrebbe essere questo il tuo caso. Semplicemente volevo metterti in guardia anche su questo aspetto ;)

VB.NET: problemi con il datagrid

Ho un problema: in alcune colonne di un datagrid devo inserire/ modificare dati, quale evento (se c'e') intercetta ogni volta che inserisco un carattere. Ho visto gli eventi key_press key_up key_dowm ma funzionano solo allínserimento del primo carattere quando il datagrid ottiene il focus
Grazie ciao

ap2000

Risponde amdbook

Se ho ben capito il tuo problema..in questo modo dovresti risolverlo...

```
'Controlli necessari
'Aggiungere alla WindosForm un oggetto
        DataGrid (DG)
Private Dt As New DataTable("Prodotti")
Private Sub Form1_Load(ByVal sender As
  System.Object, ByVal e As System.EventArgs )
        Handles MyBase.Load
'Crea un DataTable con due colonne e
        l'associa al DataGrid
CreaDt()
'Creo un oggetto DataGridTableStyle
Dim Ts As New DataGridTableStyle()
Ts.MappingName = Dt.TableName
'Creo due DAtaGridTextBoxColumn (Codice e
        descrizione)
Dim TextBoxCol As New
        DataGridTextBoxColumn()
TextBoxCol.MappingName = "Codice"
TextBoxCol.HeaderText = "CODICE"
TextBoxCol.Width = 100
'Associo l'evento al suo gestore
AddHandler TextBoxCol.TextBox.
        TextChanged, AddressOf TestoChange
Ts.GridColumnStyles.Add(TextBoxCol)
TextBoxCol = New DataGridTextBoxColumn()
TextBoxCol.MappingName = "Descrizione"
TextBoxCol.HeaderText = "DESCRIZIONE"
TextBoxCol.Width = 200
'Associo l'evento al suo gestore
AddHandler TextBoxCol.TextBox.
        TextChanged, AddressOf TestoChange
Ts.GridColumnStyles.Add(TextBoxCol)
DG.TableStyles.Add(Ts)
End Sub
Private Sub CreaDt()
Dim Col As DataColumn
Col = New DataColumn("Codice")
Dt.Columns.Add(Col)
```

```
Col = New DataColumn("Descrizione")
Dt.Columns.Add(Col)
DG.DataSource = Dt
End Sub
'Gestore dell'eventi TextChanged
Private Sub TestoChange(ByVal sender As
                    Object, ByVal e As EventArgs)
Dim T As TextBox
T = CType(sender, TextBox)
MsgBox(T.Text)
End Sub
```

Simulazione del tempo in Java

Ciao, ripropongo un quesito che pensavo fosse risolto. Devo far vedere in una simulazione lo scorrere del tempo reale e simulato (più veloce di quello reale). Devo praticamente cronometrare una determinata simulazione. Faccio coincidere con l'inizio dei tempi un determinato istante e calcolo il tempo trascorso sottraendo a quel valore il tempo di sistema. Fin qui nessun problema se non un fastidioso incremento di un'ora a causa del fatto che per java l'ora zero in Italia coincide all'una per via del meridiano. Ciò non mi interessa perchè ho tolto 3600000 millisecondi e il problema è stato risolto. Il vero scoglio è il tempo simulato. Pensavo di far coincidere 1 secondo reale con 1 minuto simulato. Per farlo ho detto: basta moltiplicare il tempo reale per una costante K=60. Invece no. Così facendo risulta che la simulazione parte dalle 13:00:00. La cosa buffa è che nelle successive moltiplicazioni rimane sempre alle 13 incrementandosi di quei minuti che ipotizzavo: 13:00:01, 13:00:02, ecc. Come è possibile? Ecco la porzione di codice:

```
Action updateCursorAction = new
                        AbstractAction(){
private Time time=new Time(0);
private Time simtime=new Time(0);
private int SIMKCOEFF = 60; //coefficiente
     K di moltiplicazione del tempo reale per
                  ottenere il tempo simulato
private int GMT = 3600000;
public void actionPerformed(ActionEvent e) {
time.setTime(System.currentTimeMillis(
                      )-startTime-GMT);
realtime.setText(time.toString());
simtime.setTime(time.getTime()
                             *SIMKCOEFF);
simulatedtime.setText(simtime.toString());}
};
//ogni 5 secondi aggiorna il tempo trascorso
time= new Timer(5000, updateCursorAction);
time.start();
```

realTime non deve visualizzare l'ora attuale, ma da quanto tempo sta girando il programma. Deve fungere da cronometro. Ecco perchè nel mio codice sottraggo al *System.currentTimeMillis()* il valore *startTime. StartTime* ha preso il valore (in millisecondi) dell'ora nel quale ha cominciato a girare il programma. Il risultato deve essere del tipo:

Tempo reale: 00:00:00
Tempo simulato: 00:00:00
dopo 5 secondi
Tempo reale: 00:00:05
Tempo simulato: 00:05:00
e così via...

Ripeto, devo usare due variabili. Una chiamata realTime che cronometra da quanto tempo sta girando il programma. L'altra, *simTime*, che simula il trascorrere del tempo in maniera + veloce (1 secondo = 1 minuto). Il problema è che per far risultare 00:00:00 devo togliere forzosamente un'ora perché essendo *realTime* e *simTime* istanze di *Time*, l'ora è riferita al meridiano di Greenwich e quindi viene commutata nell'ora italiana ovvero 01:00:00.

brehme77

Risponde BenHur73

Il problema è che Time usa un oggetto di tipo Date, quindi rappresenta a tutti gli effetti il numero di millisecondi che sono passati dalla mezzanotte del 1/1/1970. Per fare quello che dici, quindi, avresti bisogno di quella che in C si chiama *TimeSpan* (intervallo di tempo). Ho scritto una mini classe (hhmmss) che puoi usare e modificare a piacimento. Prova questo e fammi sapere se va meglio:

```
code:
  public class hhmmss
  { public hhmmss(long seconds)
    { iSeconds = seconds;
    }
  public void setTime(long seconds)
  { iSeconds = seconds;
  }
  public String toString()
  { long hh;
    int mm;
    int ss;
    long remainder;
    ss = (int)(iSeconds % 60);
    remainder = iSeconds-ss;
    mm = (int) ((remainder % 3600) / 60);
    hh = remainder / 3600;
    return new String(hh+":"+mm+":"+ ss);
  }
  private long iSeconds;
  }
  ///////////////////////////
  import java.lang.*;
  import java.io.*;
  import java.sql.*;
  public class TestTime
  { public static void main(String s[])
    { Time startTime = new Time(
                 System.currentTimeMillis());
      Time currentTime = new Time(0);
  long elapsedTime;
  hhmmss realTime = new hhmmss(0);
  hhmmss simulTime = new hhmmss(0);
  final int COEFF = 60;
  while (true)
  { currentTime.setTime(
                 System.currentTimeMillis());
  elapsedTime = (currentTime.getTime() -
                startTime.getTime()) / 1000;
  realTime.setTime(elapsedTime);
  simulTime.setTime(elapsedTime * COEFF);
  System.out.print("Start: " +
                       startTime.toString());
  System.out.print(" Current: " +
                        realTime.toString());
  System.out.println(" Simul: " +
                     simulTime.toString());}
  }
}
```

Ciao.

Ingressi analogici nel pc

Ciao a tutti. Qualcuno ha idea di come posso recuperare 3 segnali analogici (2 4-20mA e 1 Termocoppia K) e portarli in un pc senza passare per plc. Esistono schede che convertono questi segnali in seriale o altro? Vi ringrazio

Alessandro

Risponde Luca Spuntoni

Gentile Alessandro, questo può essere un ottimo 'spunto' per un articolo futuro per la rubrica elettronica. Senza ricorrere a PLC, utilizzerei un convertitore Analogico/Digitale, collegato ad un multiplexer, per ottenere la lettura di 16 segnali analogici. Occorrerebbe fissare alcune specifiche quali:

- Velocità di campionamento
- Risoluzione di campionamento
- Campo di lettura dei segnali, oltre ad altri che possono influire sulle caratteristiche di progetto.

PER CONTATTARCI
e-mail: ioprogrammo@edmaster.it
Posta: Edizioni Master,
Via Cesare Correnti, 1 - 20123 Milano